CENTRO PAULA SOUZA
FACULDADE DE TECNOLOGIA
FATEC SANTO ANDRÉ

Tecnologia em Eletrônica Automotiva

BRUNO ZANI SAMPAIO

ROBERTO JUNCKER

PLATAFORMA DIDÁTICA PARA AQUISIÇÃO DE DADOS VIA INTERFACE USB E *LabVIEW* ® COM BOOTLOADER INTEGRADO

Santo André - São Paulo
2014

CENTRO PAULA SOUZA
FACULDADE DE TECNOLOGIA
FATEC SANTO ANDRÉ

Tecnologia em Eletrônica Automotiva

BRUNO ZANI SAMPAIO

ROBERTO JUNCKER

PLATAFORMA DIDÁTICA PARA AQUISIÇÃO DE DADOS VIA INTERFACE USB E *LabVIEW* ® COM BOOTLOADER INTEGRADO

Trabalho de Conclusão de Curso entregue à Fatec Santo André como requisito parcial para obtenção do título de Tecnólogo em Eletrônica Automotiva.

Orientador: Prof. Dr. EDSON CAORU KITANI
Co-orientador: Prof. WESLLEY MEDEIROS TORRES

Santo André - São Paulo
2014

FICHA CATALOGRÁFICA

CUTER (Biblioteca FATEC)

Juncker, Roberto

Plataforma didática para aquisição de dados via interface USB e *LabVIEW* ® com Bootloader integrado / Roberto Juncker, Bruno Zani. – Santo André, 2014. – 87 f:il.

Trabalho de conclusão de curso – FATEC – Santo André.

Curso de Eletrônica automotiva, 2014.

Orientador: Prof. Dr. Edson Kaoru Kitani

1. USB 2. *LabViEW* 3. PIC 4. *Bootloader*.
I. Sampaio, Bruno Zani
II.Título: Plataforma didática para aquisição de dados via interface USB e *LabVIEW* ® com Bootloader integrado

CDD (Biblioteca FATEC)

Fatec Centro Paula Souza

Faculdade de Tecnologia de Santo André

GOVERNO DO ESTADO SÃO PAULO

LISTA DE PRESENÇA

SANTO ANDRÉ, 06 DE DEZEMBRO DE 2014.

LISTA DE PRESENÇA REFERENTE À APRESENTAÇÃO DO TRABALHO DE CONCLUSÃO DE CURSO COM O TEMA **"PLATAFORMA DIDÁTICA PARA AQUISIÇÃO DE DADOS VIA INTERFACE USB E LABVIEW COM BOOTLOADER INTEGRADO"** DOS ALUNOS DO 6º SEMESTRE DESTA U.E.

BANCA

PRESIDENTE:

PROF. DR. EDSON CAORU KITANI _____

MEMBROS:

PROF. [redacted]. WESLLEY M. TORRES _____

ENGª. SRTA LARISSA L. RESENDE _____

ALUNO:

BRUNO ZANI SAMPAIO Bruno Zani Sampaio

ROBERTO JUNCKER Roberto Juncker

DEDICATÓRIA

Dedico este trabalho aos meus pais, Antonio e Diva, que são o alicerce do meu desenvolvimento e também à minha esposa, pois sem ela este trabalho e muitos dos meus sonhos não se realizariam.

Bruno Zani Sampaio

Dedico este trabalho ao meu pai, Mario Alberto Juncker (*in memoriam*), a minha mãe, Dalva Juncker, ao meu querido irmão, Ricardo Juncker (*in memoriam*) e a minha querida irmã, Sandra Juncker, os quais sempre me incentivaram e apoiaram.

Roberto Juncker

AGRADECIMENTOS

Agradecemos a todos os professores, mestres, doutores e colaboradores da Faculdade de Tecnologia Santo André, bem como aos nossos amigos e familiares, que sempre nos incentivaram e estiveram dispostos a nos ajudar, contribuindo direta ou indiretamente para a realização deste trabalho.

“O sucesso nasce do querer, da determinação e da persistência em se chegar a um objetivo. Mesmo não atingindo o alvo, quem busca e vence obstáculos, no mínimo fará coisas admiráveis.”

(José de Alencar)

RESUMO

A placa didática elaborada neste trabalho consiste em um protótipo que faz uso da porta de comunicação USB para realizar a integração do *hardware* com o *firmware* do projeto, por meio da transferência de dados entre um microcontrolador PIC 18F4550 da Microchip e a ferramenta computacional LabVIEW, baseada na linguagem G (linguagem gráfica de programação). Todo este trabalho possui o intuito de contribuir com a ampliação do conhecimento de futuros alunos ou pessoas que adquirirem interesse em realizar estudos e trabalhos posteriormente com esta plataforma didática. O código do microcontrolador foi escrito em linguagem de programação C e o compilador utilizado foi o CCS 4.140.30.19. Com o LabVIEW foi realizado a programação gráfica necessária para estabelecer comunicação, por meio de uma função chamada *Call Library Function Node,* que faz o uso adequado da *library* mpusbapi, disponibilizada pela Microchip. A ponte que estabelece comunicação entre o microcontrolador PIC18F4550 e o PC é feita através do *driver* mchpusb da Microchip e as *libraries* USB do *software CCS C Compiler*. O aplicativo utilizado para executar a programação de nosso *firmware* via *Bootloader* foi o *HID Bootloader* criado pela Microchip. O tipo de comunicação USB adotado foi a transferência do tipo *bulk,* devido ao fato deste método permitir envios de grandes quantidades de dados, assim como também garantir a integridade dos mesmos. Com o protótipo apresentado neste trabalho é possível, em teoria, receber ou enviar pacotes de dados de informações com uma velocidade de até 12 Mbps.

Palavras chaves: USB, LabVIEW, Mpusbapi.dll, Bootloader, *Mchpusb*, Microcontrolador PIC, Aquisição de Dados.

ABSTRACT

The didactic board developed in this work is a prototype that uses the USB communication port for the integration of the project's hardware with the firmware, through the data transfer between a microcontroller PIC18F4550, from Microchip and LabVIEW software tool, held based on G language (graphic programming language). The microcontroller code was written in C programming language and the compiler used was the CCS 4.140.30.19. With LabVIEW was done the graphical programming necessary to establish communication, through a feature called Call Library Function Node, which makes proper use of the library mpusbapi which is provided by the Microchip. The bridge that establishes communication between PIC18F4550 microcontroller and the PC is made through mchpusb Microchip driver and the USB libraries from the CCS C Compiler software. The application used to perform the programming of the Bootloader's firmware was the HID Bootloader created by Microchip. The USB communication type used in the project was the transfers of bulk type, due to the fact of this method allow sending large amounts of data, as well ensuring their integrity. With the prototype presented in this paper is possible, in theory, to receive or send data packets of information with a speed of up to 12Mbp/s.

Key words: USB, LabVIEW, Mpusbapi.dll, Bootloader Mchpusb, Microcontroller PIC, Data Acquisition.

SUMÁRIO

Lista de Figuras

Lista de Abreviaturas

ADC	*Analog to Digital Converter* (Conversor Analógico-Digital)
CCS	*Custom Computer Service*
CD	*Compact Disc (*Disco Compacto)
CCP	*Capture Compare PWM*
CDC	*Communications Device Class* (Classe de Comunicação do Dispositivo)
DLL	*Dynamic Link Library*
DVD	*Digital Versatile Disc* (Disco Digital Versátil)
ECU	*Eletronic Control Unit* (Unidade de Controle Eletrônico)
GND	*Ground* [terra (eletricidade)]
HID	*Human Interface Device* (Dispositivo de Interface Humana)
IBM	*International Business Machines*
I/O	*In/Out (*entrada ou saída de um módulo eletrônico)
KB	*Kilobytes* (Quilobytes)
Kbps	*Kilo bits per second* (Kilobits por segundo)
LabVIEW	*Laboratory Virtual Instruments Engineering Workbench*
LED	*Light Emitting Diode* (Diodo Emissor de Luz)
Mbps	*Mega bits per second* (Megabits por Segundo)
MCLR	*Master Clear* (Reset do PIC)
MHz	Unidade de frequência derivada do SI (Mega-hertz)
MSD	*Mass Storage Device* (Dispositivo de Armazenamento em Massa)
NRZI	*Non Return to Zero Inverted*
PC	*Personal Computer*
PID	*Product ID*- Número de Identificação do Dispositivo USB
PIC	*Programmable Integrated Circuit* (Circuito Integrado Programável)
PLL	*Phase-locked loop*
PnP	*Plug and Play* (Ligar e Usar)
PWM	*Pulse Width Modulation* (Modulação por largura de pulso)
STR	Sistema de Tempo Real
USB	*Universal Serial Bus*
V	Unidade de tensão derivada do SI (volt)
VI	*Virtual Instrument* (Instrumento Virtual)
VID	*Vendor ID* – Identificação do Fabricante do Dispositivo USB

1 Introdução

"Em última análise, o cerne da economia não é a tecnologia, seja um microchip ou a rede de comunicação global. O cerne da economia é a mente humana."

Alan M. Webber

A aquisição de dados é uma forma de obter informações do mundo real com o objetivo de criar dados que possibilitem a sua manipulação por um computador. Geralmente esses sistemas são compostos por sensores que são capazes de converter determinados parâmetros medidos em um sinal eletrônico, permitindo assim o monitoramento, análise e manipulação dos dados adquiridos. Essas ações realizadas com os dados normalmente são realizadas por meio de *softwares* iterativos e infindáveis códigos em linguagem de programação.

Olhando por outro lado, o próprio ser humano possui um complexo sistema de aquisição de dados. Estamos continuamente recolhendo informações, do ambiente em que vivemos, como a luz, som, odores, sabores e outras muitas sensações. Baseamo-nos nestas amostras de dados para efetuar as tomadas de decisões. Da mesma maneira que os seres humanos, muitas aplicações realizam ações ou tomadas de decisões para controlar, corrigir ou manipular o sistema em causa de acordo com as informações coletadas, sendo que essas aplicações não se restringem somente à aquisição de dados, mas cada elemento funcional do sistema tem papel fundamental para efetuar a coleta de informações ou atuar conforme o objetivo estabelecido no início de um determinado projeto.

1.1. Motivação

Atualmente muitas empresas fabricam "kits" e plataformas didáticas USB, sendo que a popularidade desses sistemas aumenta ainda mais com a crescente e contínua evolução da eletrônica e dos computadores. Entre outras razões para essa popularidade, destacam-se o baixo custo, a flexibilidade, o monitoramento em tempo real de variáveis de um determinado processo (temperatura, luminosidade, pressão, entre outros.) e, também, a possibilidade de controlar atuadores por meio de um computador tipo PC (*Personal Computer*).

Olhando em paralelo a este cenário, a possibilidade de confeccionar uma plataforma didática similar de autoria própria e que ainda poderá ser utilizada por outros alunos futuramente é uma grande motivação para levarmos este projeto adiante.

1.2. Objetivos

Baseado nas motivações discutidas na subseção 1.1, entende-se como objetivo principal deste trabalho o desenvolvimento de uma plataforma didática com *Bootloader* integrado, fazendo o uso da porta de comunicação USB (*Universal Serial Bus*) para que seja realizado a integração do *hardware* com o *firmware* do projeto, por meio da transferência de dados entre um microcontrolador PIC 18F4550 da Microchip e a ferramenta computacional LabVIEW, baseada na linguagem G (linguagem gráfica de programação), podendo assim coletar e gerar certos tipos de sinais com algumas limitações que serão discutidas no capítulo 6 .

1.3. Contribuições

As contribuições estão relacionadas com os objetivos descritos na subseção 1.2 e são:

a) Uma plataforma didática que possui fácil manipulação da estratégia de comunicação, ou seja, tanto do *hardware*, do *firmware*, como também do *driver* USB, que realiza a integração do sistema com a ferramenta computacional LabVIEW;
b) Espera-se como resultado deste presente trabalho uma plataforma didática capaz de acelerar o aprendizado dos alunos da própria FATEC ou de outras pessoas que ficarem interessadas pelo projeto, contribuindo assim para o desenvolvimento de projetos pertencentes à grade curricular da disciplina Ferramentas Computacionais ou para a ampliação do conhecimento de terceiros interessados no trabalho;
c) Uma plataforma didática com um baixo custo de confecção.

1.4. Organização do Trabalho

Este trabalho está organizado da seguinte forma: o capítulo 2 trata de uma revisão da fundamentação teórica necessária para o desenvolvimento desta plataforma didática, tendo como foco principal a tecnologia USB e conceitos de programação estruturada de instrumentos virtuais através do LabVIEW. O capítulo 3 traz de forma sucinta a parte conceitual sobre sistemas de aquisições de dados e suas principais características. O capítulo 4, por sua vez, trata sobre os conceitos de *firmware* e de *hardware* necessários para a implantação do *Bootloader* tanto neste projeto como em alguma outra aplicação plausível qualquer.

No capítulo 5 são descritos os processos e etapas que foram necessárias para a criação desta plataforma didática, sendo descritos tanto a parte de programação em linguagens C e G, como a metodologia usada para a montagem do *hardware* da plataforma didática USB. Neste capítulo também pode ser observado os resultados obtidos ao longo de todo o processo de desenvolvimento. Por fim, no capítulo 6, apresentam-se as conclusões obtidas com o desenvolvimento deste projeto, assim como as dificuldades encontradas ao longo do percurso como também as possibilidades de aplicações e propostas futuras para novos desafios.

2 Fundamentação Teórica

"O conhecimento pode ajudar ou inibir a inovação, mas, quando vários conhecimentos se misturam, aí a inovação acontece."

Dorothy Leonard

Será apresentado neste capítulo alguns conceitos e fundamentos teóricos sobre tecnologia USB e a ferramenta computacional LabVIEW, assim como uma breve revisão bibliográfica dos mesmos.

2.1 A Tecnologia USB

A comunicação serial USB foi uma das maiores inovações tecnológicas já realizadas para os computadores PC's padrão IBM, tendo como principal objetivo atender as necessidades de fabricantes de periféricos que buscavam um modo de comunicação com custo reduzido, com conectores universais e de fácil instalação (ULLRICH, 2009). Foi a partir desta necessidade de comunicação única que um conjunto de empresas desenvolveu o protocolo de comunicação USB (STALLINGS, 2002).

Antigamente, a instalação de periféricos em um computador era uma tarefa muito complicada, pois havia a necessidade de abertura da máquina pelo usuário. A chegada do padrão de conector PnP (*Plug and Play*) tornou possível a instalação de novos periféricos no PC por usuários inexperientes, possibilitando o uso dos periféricos imediatamente após a instalação. Desse modo ficou mais fácil para o usuário final realizar a instalação de um periférico USB, não sendo necessário para isso desligar o PC. Em contrapartida as dificuldades aumentaram para os fabricantes criarem dispositivos que utilizem este tipo de tecnologia (ULLRICH, 2009).

2.1.1 Principais Vantagens do Padrão USB

Existem várias vantagens na utilização de conectores com o padrão USB, sendo elas:

a) Conexão PnP, sendo que quase todos os dispositivos USB foram criados para serem conectados ao PC e serem utilizados logo em seguida. No caso deste projeto é exigida a instalação de um *driver* inicialmente para que a comunicação seja estabelecida. Mais detalhes podem ser observados na seção 5.4.

b) Na alimentação elétrica não é exigida outra fonte de energia além da fornecida pela própria conexão;
c) A compatibilidade com diversas plataformas e sistemas;
d) O conector USB é do tipo *hot-swappable* (conectáveis "a quente"), permitindo a conexão ou desconexão de dispositivos USB a qualquer momento sem a necessidade de desligar ou reiniciar o PC. Podem ser conectados até 127 dispositivos ao host, diretamente ou através de hubs USB (ALECRIM, 2009).
e) Flexibilidade, sendo suportado um grande número de pacotes de dados de tamanhos variados e o controle da capacidade do fluxo de dados é realizado internamente no protocolo USB (COMPAQ CORPORATION, 2008).

2.1.2 Topologia e Tipos de Conectores USB

A topologia USB compreende uma forma denominada *tiered star* (estrela em camadas). Nesta topologia um único computador *(root)* pode realizar conexão com vários periféricos, assim como *hubs* (outros pontos de conexão para outros periféricos). No centro de cada estrela há um *hub* e cada ponto é um dispositivo que se conecta à porta de um *hub* que normalmente possui duas, quatro, ou sete portas (ZEMBOVICI & FRANCO, 2009).

A função (*function*), pode ser um dispositivo ligado ao *hub* principal ou externo, provendo assim certa capacidade ao *host*, como pode ser observado na Figura 1 abaixo.

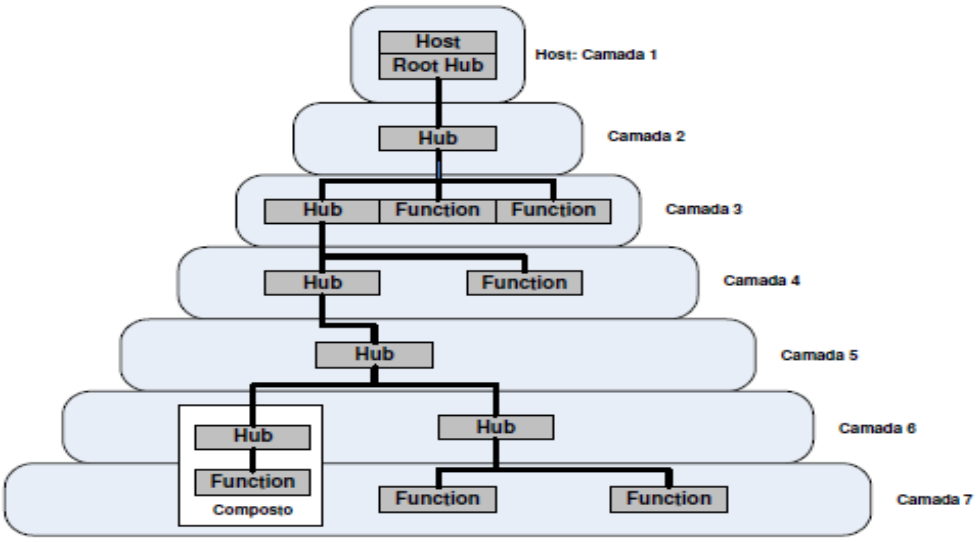


Figura 1 - Topologia de um sistema USB [Extraída de (AMORIM, 2008)].

Já os tipos e padrões de conectores USB podem ser observados na Figura 2:

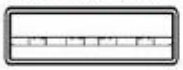
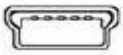
Tipo	Imagen da porta	Imagen do conector
Tipo A	4,5mm x 12mm 	
Tipo B	7,3mm x 8,5mm 	
Mini-A	3mm x 6,8mm 	
Mini-B	3mm x 6,8mm 	

Figura 2 - Tipos de conectores USB [Extraída de (CEBRAC, 2012)].

2.1.3 A Interface Física USB

O cabo USB 2.0 é constituído por quatro cabos internos (dois pares de cabos). Um par é trançado e formado por um fio branco e outro verde (D+ e D-), ou verde e amarelo, variando de acordo com o tipo de fabricante. O outro par é constituído pelos fios V+, com a coloração vermelha e o GND, com a cor preta ou marrom. Esses dois pares de fios são envolvidos por uma camada de alumínio e também blindados com uma pequena malha externa de cobre. Por fim há uma capa protetora constituída de plástico. Todas estas características podem ser observadas na Figura 3.

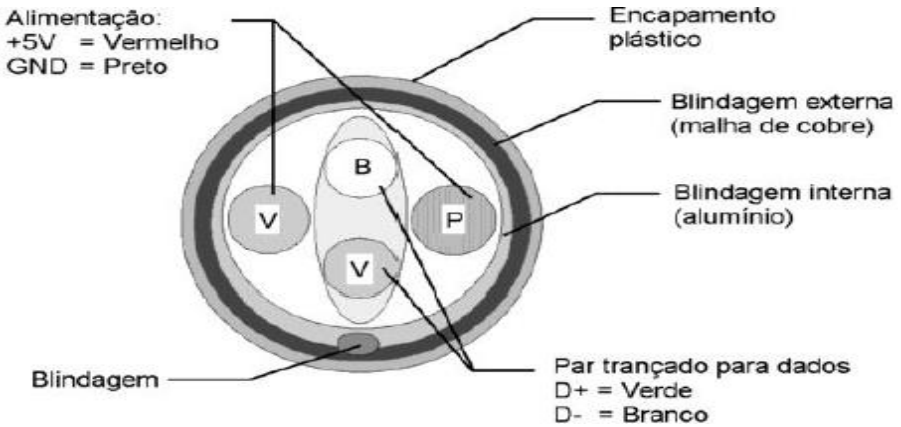


Figura 3 - Tipos de conectores USB [Extraída de (CEBRAC, 2012)].

Através destes cabos é gerado um tipo de sinal diferencial, em que a transferência de dados é realizada de forma bidirecional, porém não simultaneamente. A transferência é realizada ponto a ponto nos fios D+ e D-, e a codificação dos dados é do tipo NRZI (*Non Return to Zero Inverted*), mantendo um pulso de tensão constante durante a transmissão de um intervalo de bit. Neste tipo de codificação diferencial, os sinais são decodificados pela comparação da polaridade do símbolo adjacente, ou seja, o nível lógico zero faz com que o sinal não comute de estado e o nível lógico um gera uma transição no estado do sinal atual. Já os cabos V+ e GND são responsáveis por fornecer a energia para os dispositivos (SOUZA, 2010).

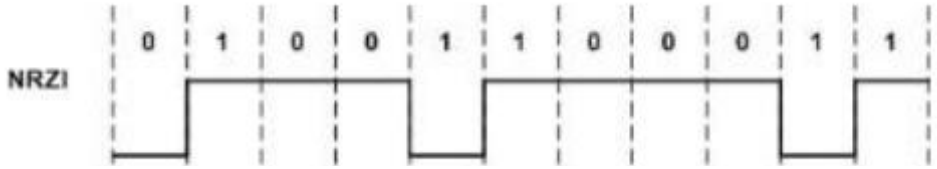


Figura 4 - Codificação NRZI [Extraída de (FILHO, 2005)].

Neste mesmo sinal diferencial entre D+ e D-, que também possui alta imunidade a ruídos e interferências elétricas, são enviados e recebidos pacotes de dados, podendo ser observado o sinal *Sync* (sincronização), PID (*Productor ID)* e uma parte opcional. Estes três sinais compõem o pacote onde está a informação que se deseja transmitir. Tudo isso é representado na Figura 5:

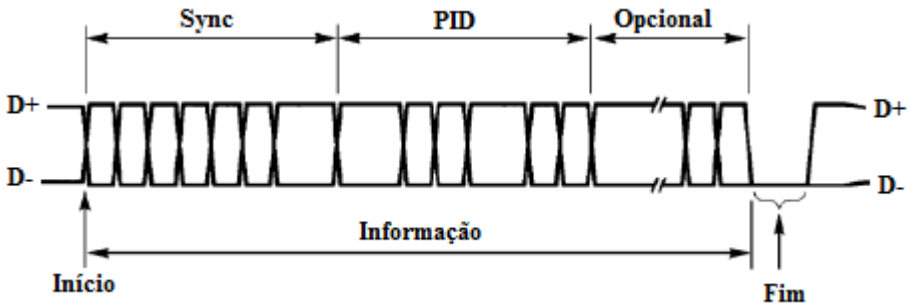


Figura 5 - Composição de um pacote de dados [Adaptada de (CUETO & ESTRADA, 2009)].

2.1.4 Modelo de Fluxo de Dados e Tipos de Transferência USB

O modelo de fluxo de dados é realizado entre o *software* do *host* e um determinado *endpoint* (ponto final) de um dispositivo USB. Esse "envolvimento" das duas partes é denominado *pipe* (canal) e normalmente o fluxo de dados em um *pipe* é independente do fluxo de dados em outro *pipe*. O *endpoint* nada mais é do que um *buffer*, ou seja, uma área de

memória reservada no dispositivo para armazenar os dados que trafegam em um *pipe* (MESSIAS, 2013). Um dispositivo USB possui vários *endpoints*, devendo ter um *endpoint* que suporta o canal para que seja realizado o fluxo de dados para o dispositivo e outro *endpoint* para suportar o fluxo de dados no sentido oposto (ZEMBOVICI & FRANCO, 2009).

A transferência de dados via USB deve ser escolhida de acordo com a aplicação desejada e ser configurada corretamente para um uso específico. Esta escolha deve ser baseada nas características dos quatro tipos de transferência USB existente, sendo que as características principais desses tipos são descritas abaixo:

- Transferência de controle: usada para configurar um dispositivo USB no instante em que a conexão é efetuada, sendo disponibilizado neste instante os descritores e os parâmetros de configuração. Outros *drivers* podem ser escolhidos para utilizar esta transferência e ela pode ser usada de modo a controlar outros *pipes* no dispositivo (ZEMBOVICI & FRANCO, 2009).
- Transferência do tipo *bulk*: é um tipo de transmissão não periódica que não possui latência garantida e é utilizada por dispositivos que trabalham com um alto volume de dados, mas que consegue garantir a integridade dos mesmos (AMORIM, 2008).
- Transferência do tipo interrupção: é uma transmissão de baixa frequência geralmente utilizada por dispositivos que lidam com uma baixa transferência de dados, como por exemplo, o *mouse.* Nesse tipo de transferência os dados devem ser disponibilizados em um tempo limite (ALECRIM, 2009) e (ZEMBOVICI & FRANCO, 2009).
- Transferência do tipo isossíncrona: essa classe de transferência deve ser aplicada em transmissões contínuas, onde não há detecção de erros para não atrasar a comunicação, sendo que o tempo é uma restrição para a mesma. A transmissão isossíncrona envolve priorização de largura de banda e um fluxo de dados constante. Um exemplo deste tipo de transferência é a comunicação de áudio em tempo real utilizada por certos dispositivos como alto-falantes (AMORIM, 2008).

Outro ponto importante é que o tamanho do *endpoint* é dependente do tipo da transferência USB e da velocidade (*full* ou *low speed*), sendo que a velocidade *low speed* não pode ser estabelecida pelas transferências *bulk* e isossíncrona. (Microchip, 2006).

Mais detalhes podem ser observados na Figura 6 abaixo:

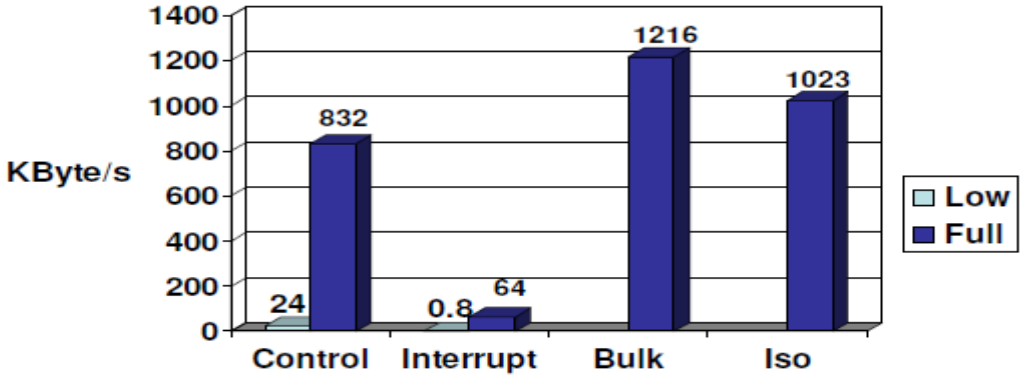


Figura 6 - Tipos de transferência de dados e suas taxas máximas de transferência teórica por e*ndpoint* em *low* e *full-speed* [Extraída de (11ºSeminário Técnico Microchip Masters Brasil, 2006)].

2.1.5 Hierarquia de Descritores

Todos os dispositivos USB possuem uma hierarquia de descritores cujo papel é informar ao *host* quais as características de funcionamento do dispositivo conectado (SANTOS, 2009). A hierarquia de descritores é constituída por quatro níveis, como pode ser observado na Figura 7 abaixo:

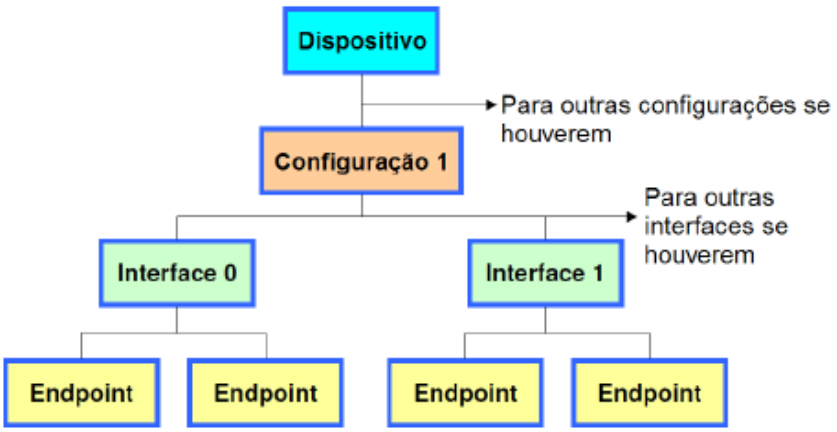


Figura 7 - Hierarquia dos descritores USB [Adaptado de (11ºSeminário Técnico Microchip Masters Brasil, 2006)].

O primeiro nível da hierarquia é formado pelo descritor de dispositivo, que é incumbido de informar ao *host* as características do dispositivo, qual o PID (*Product ID*) e VID (*Vendor ID*) do dispositivo, o número de série, a classe e outros mais. É importante salientar que um dispositivo USB pode possuir somente um descritor de dispositivo. É através do descritor de dispositivo que o *host* verifica qual versão de USB o dispositivo suporta.

O segundo nível é constituído pelos descritores de configuração, sendo estes os portadores de informações a respeito da capacidade funcional do dispositivo como: o tipo de alimentação de energia, se esta alimentação é própria ou não (*self powered* ou *bus-powered*), a corrente elétrica consumida, entre outros (MESSIAS, 2013). Sobre a hierarquia dos níveis há os descritores de *interface* que possuem informações de relação do dispositivo com o *endpoint*.

Já os descritores de *endpoint*, por sua vez, informam o *host* sobre um *pipe* específico de comunicação (DOGAN, 2008).

2.1.6 As Classes USB dos Dispositivos

O protocolo USB define códigos de diversas classes de dispositivos para carregar o *driver* e também identificar a sua funcionalidade. Dessa maneira permite-se que cada codificador de *driver* de dispositivo seja compatível com diferentes fabricantes seguindo com o código de certa classe de dispositivo (SANTOS, 2009).

Existem diversas classes de dispositivos USB, dividindo-se basicamente entre três classes, sendo estas a classe CDC (*Comunication Device Class*), HID (*Human Interface Device*) e MSD (*Mass Storage Device*).

A classe CDC compreende um método de comunicação que pode ser utilizado por diversos dispositivos como modems, aplicações de telefonia para realizações de chamada de voz normais, fax e placas de rede.

Os dispositivos que facilitam a comunicação entre o operador e o computador fazem parte da classe HID. A instalação destes dispositivos é facilitada porque eles já possuem seu protocolo de comunicação pré-estabelecido. Podem ser citados exemplos como o *mouse*, teclado e o *joystick*.

Dispositivos que são capazes de armazenar uma quantidade massiva de dados, como leitores ou gravadores de CD e DVD pertencem à classe de dispositivos MSD.

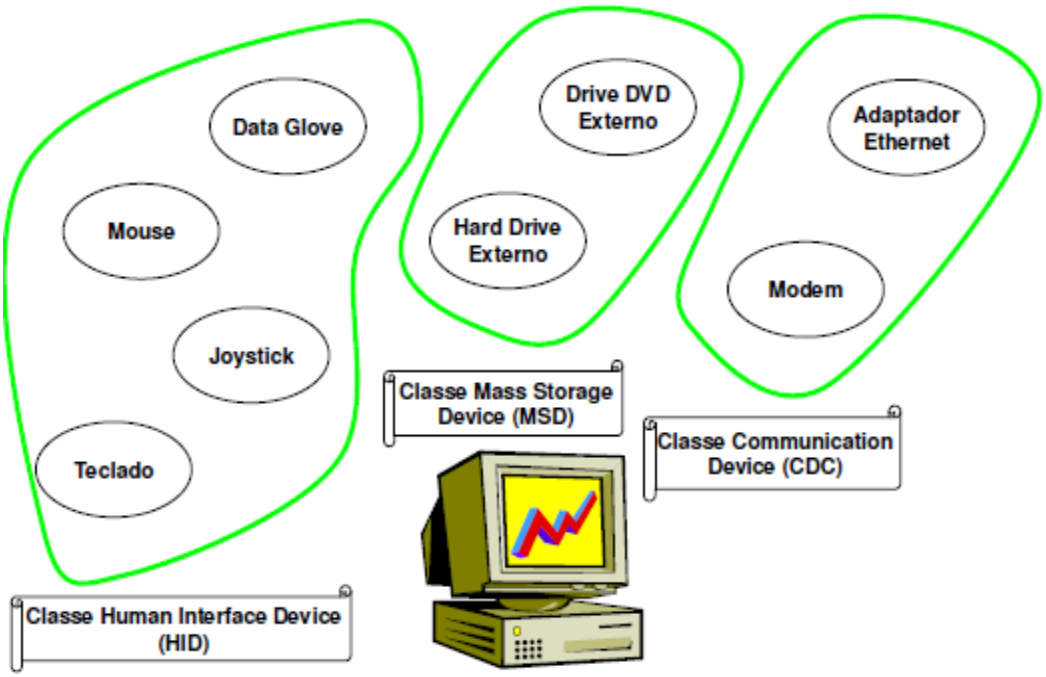


Figura 8 - Classe dos dispositivos USB [Extraída de (11ºSeminário Técnico Microchip Masters Brasil, 2006)].

2.1.7 Processo de Enumeração do Sistema USB

A Enumeração USB é um processo de detecção, em que as fases desse processo determinam o modo como o *host* reconhece os dispositivos conectados ao barramento.

Antes de iniciar a comunicação das aplicações com o dispositivo é necessário que o *host* reconheça o dispositivo e atribua um *driver* de dispositivo. A enumeração é responsável pelo cumprimento destas tarefas por meio de troca de informações. Este processo inclui a atribuição de um endereço para o dispositivo, assim como a leitura de seus descritores, atribuição e *loading* do *driver* do dispositivo. Dessa maneira seleciona-se uma configuração que especifica os requisitos de alimentação de energia do dispositivo, os *endpoints* e mais outros recursos (AXELSON, 2001).

Não é necessário possuir um conhecimento detalhado sobre o processo de enumeração para desenvolver um periférico USB, mas o conhecimento do funcionamento deste processo de uma forma geral é de extrema importância para a escrita de *firmwares* relacionados à enumeração USB.

Da perspectiva do usuário o processo de enumeração é invisível e automático, exceto quando uma mensagem surge na tela, descrevendo a detecção do dispositivo e se a sua configuração foi bem sucedida. Às vezes, em determinadas aplicações, é necessário que o usuário auxilie a enumeração do dispositivo na primeira utilização, especificando onde o *host*

deve procurar o *driver* do dispositivo ou até mesmo escolher o seu *driver*. Quando o processo de enumeração é finalizado o *Windows* adiciona este novo dispositivo no Gerenciador de Dispositivos do Painel de Controle e o remove quando o usuário remove o dispositivo do barramento. Este auxílio realizado ao usuário poderá ser visto com detalhes no capítulo 5.4, onde será apresentado de forma prática o processo de enumeração do dispositivo USB do projeto desenvolvido neste trabalho.

Tipicamente o *firmware* contém as informações que o *host* irá solicitar, sendo que uma combinação ideal de *hardware* e *firmware* decodifica e responde corretamente à solicitação feita pelo *host* (AXELSON, 2001).

As Figuras 9 e 10 ilustram a situação de um dispositivo MSC que foi enumerado e outro que não foi enumerado no Gerenciador de Dispositivos do Windows. Quando não ocorre a enumeração aparece um pequeno símbolo amarelo de exclamação no dispositivo correspondente.

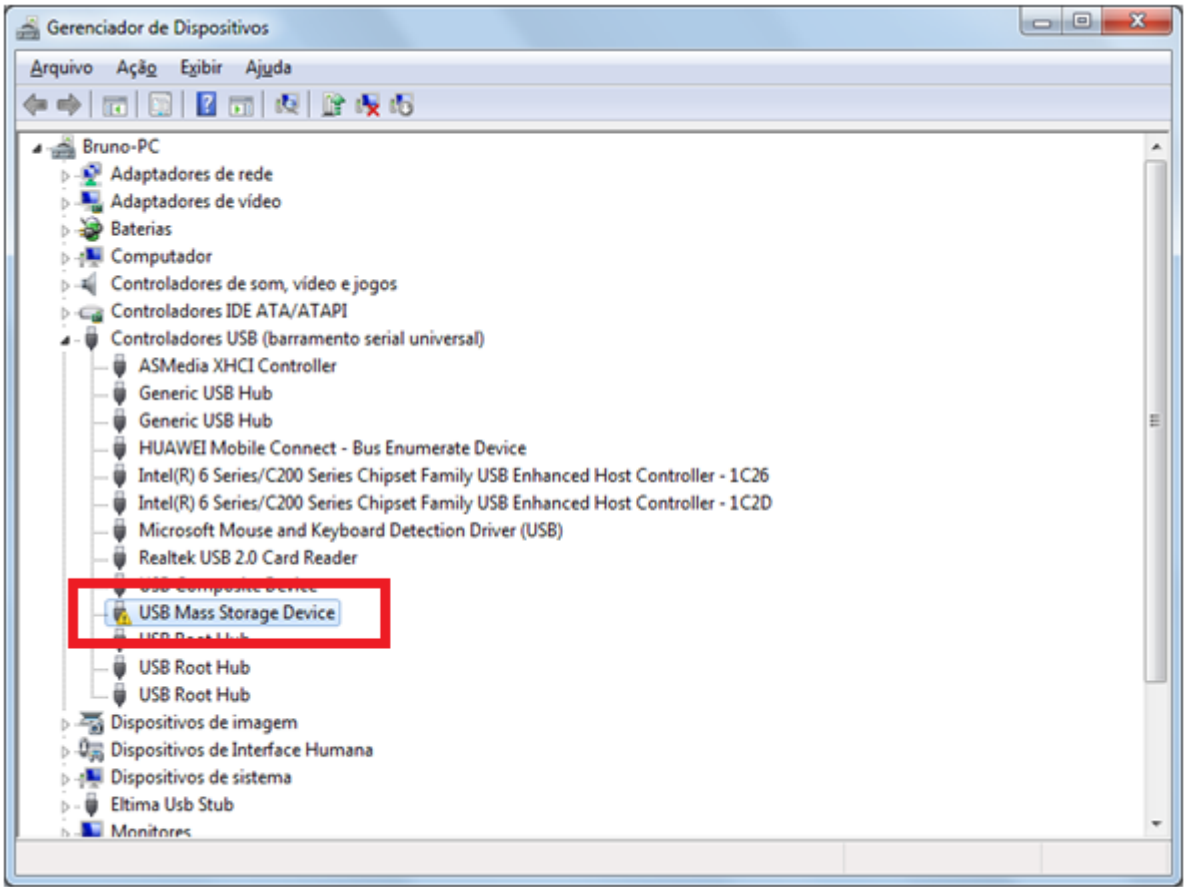


Figura 9 - Dispositivo USB não enumerado (Fonte: Os autores).

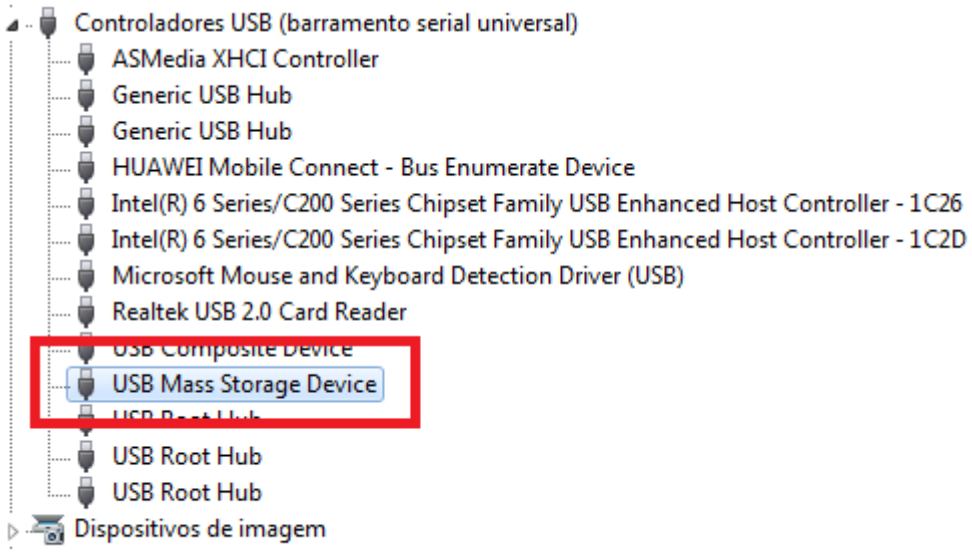


Figura 10 - Dispositivo USB enumerado (Fonte: Os autores).

2.2 A Ferramenta LabVIEW

O LabVIEW é uma ferramenta computacional que faz uso de uma linguagem gráfica de programação, com a utilização de ícones ao invés de linhas de texto, para construir aplicações. O LabVIEW não executa sua programação através de código em forma de texto, mas sim por meio de fluxo de dados, sendo que esse fluxo determina os passos da execução da aplicação. O fluxo de dados traz algumas vantagens para determinadas aplicações, principalmente as de aquisição e manipulação de dados.

Essa ferramenta disponibiliza ao usuário duas interfaces, o *Front Panel* (Painel Frontal) e o *Block Diagram* (Diagrama de Bloco). A linguagem gráfica de programação que será responsável pelo fluxo de dados do programa é realizada no *Block Diagram* e quando a VI (*Virtual Instrument*) é executada as operações da aplicação podem ser visualizadas através de indicadores no *Front Panel*, assim como podem ser controladas no mesmo (o *Front Panel* é constituído por terminais interativos de entrada e saída da VI, os controles e indicadores, respectivamente). Sendo assim, a interação com o usuário é feita no *Front Panel* através de dados adquiridos ou gerados pelo *Block Diagram.*

A Figura 11 abaixo mostra as duas interfaces de uma VI construída e executada pela ferramenta LabVIEW:

Figura 11 - Exemplo de uma janela de um painel frontal e seu diagrama de bloco correspondente (Adaptado e disponível em <http://www.docstoc.com/docs/30543443/LABVIEW-INTRODUCTION>).

2.2.1 Programação Estruturada de Instrumentos Virtuais

As VI's ou Instrumentos Virtuais do LabVIEW são assim denominados devido ao fato de sua aparência e métodos de operações possuírem grandes semelhanças com instrumentos físicos de medições, podendo ser citado exemplos como o osciloscópio e o multímetro. Cada VI possui funções que manipulam a entrada de informações pela *interface* do usuário ou outras fontes específicas, indicando a informação ou movendo-a para outros arquivos ou computadores.

A programação é realizada através do *Block Diagram* seguindo um fluxo de dados que é estruturado principalmente por meio de representações gráficas de *loops* (laços), fios, nós, terminais e funções, entre outras infindáveis representações gráficas do LabVIEW. Os nós são objetos do *Block Diagram* que têm entradas e/ou saídas que efetuam suas operações quando a VI entra em operação. Os fios realizam a transferência de dados entre esses objetos, possuindo diferentes colorações, estilos e espessuras, de acordo com o tipo de dado que será transferido. Cada fio tem uma única origem de dado. Já os terminais são utilizados para representar o tipo de dados para o indicador ou o controle. Por fim as funções são utilizadas para executar operações numéricas, conversão de dados, operações booleanas e assim por diante. A Figura 12 a seguir mostra algumas das ferramentas de programação do LabVIEW:

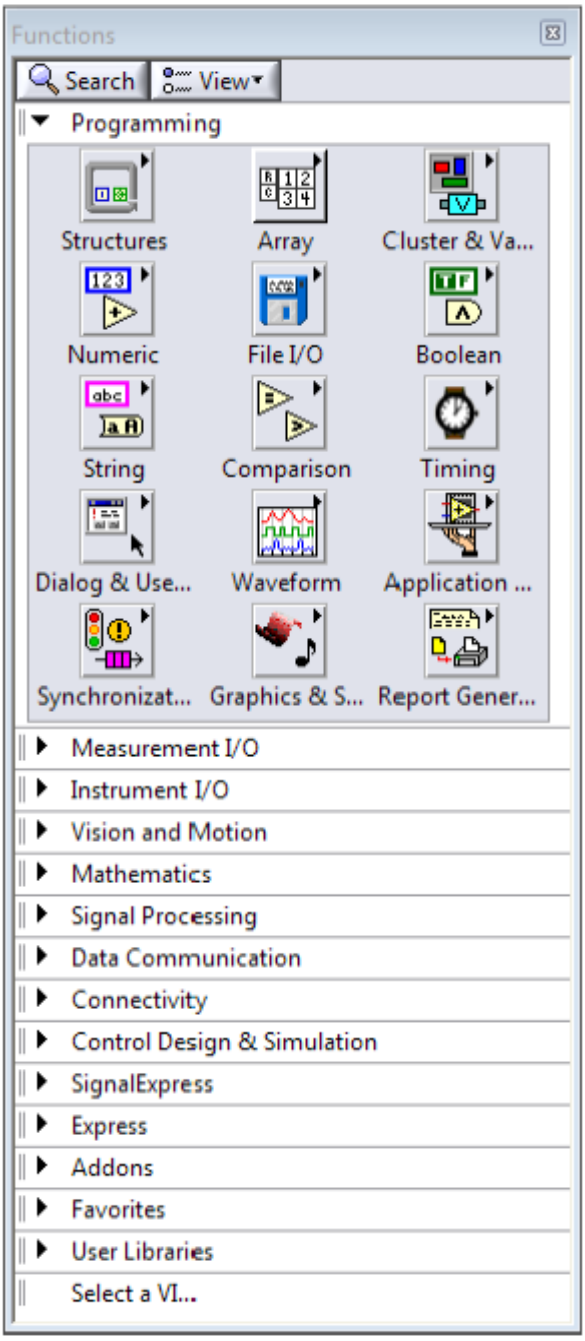


Figura 12 - Lista de ferramentas disponíveis para estruturar a programação em LabVIEW (Fonte: Os autores).

Como em todo tipo de linguagem de programação é necessário o conhecimento sobre os tipos de variáveis para estruturar corretamente o código, e no LabVIEW não é diferente. Mas como a *interface* desse *software* é visual, a diferenciação dos tipos de variáveis do programa também é realizada visualmente por meio das cores das linhas que ligam os blocos e também através das formas de seu traçado.

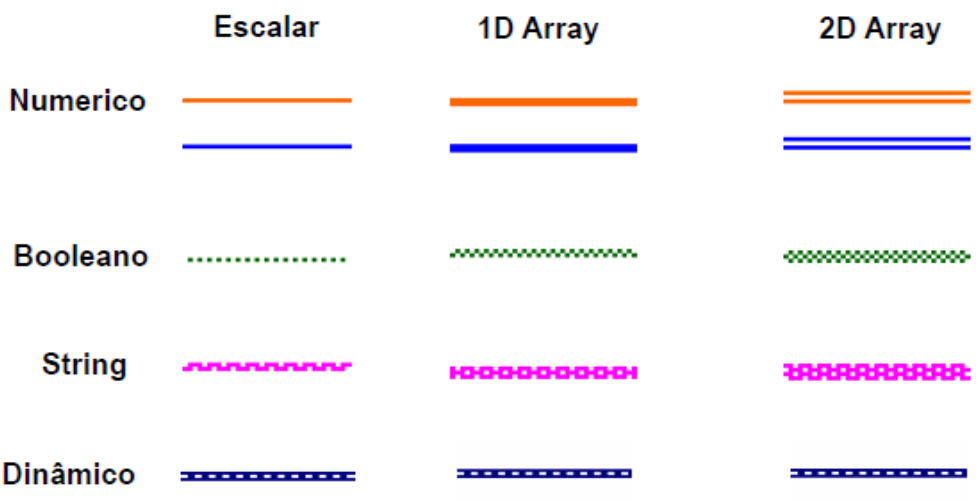


Figura 13 - Representação dos dados por cor e tipo de conexões [Adaptado de (KITANI, 2013)].

Em LabVIEW a estrutura de execução dos instrumentos virtuais é realizada com base no fluxo de dados internos, sendo que as funções aguardam até que todas as suas entradas estejam totalmente presentes para então executar os seus respectivos códigos de designação e liberar os dados calculados em suas saídas (BARBOSA, 2008).

2.2.2 Benefícios da Utilização do LabVIEW

Em relação ao desempenho, o LabVIEW possui um compilador que gera um código gráfico que é traduzido para um código de máquina executável. O código executável funciona com a ajuda de um mecanismo de tempo de execução do LabVIEW, que contém um código pré-compilado para executar tarefas comuns que são definidas pela linguagem G. Desse modo é fornecido uma interface consistente para diversos sistemas operacionais, componentes de hardware e sistemas gráficos, além de tornar possível a portabilidade do código entre outras plataformas (*LabVIEW* ® Manual, 2006).

O LabVIEW permite também a execução de tarefas em modo paralelo, sendo isto facilmente realizado por meio da utilização de dois ou mais laços *while loops,* sendo assim um grande benefício para sequências de testes, registro de dados e *interfaces* de *hardware*.

Existem também diversas opções de pacotes do LabVIEW que fornecem milhares de bibliotecas com um elevado número de funções para realizar aquisições de dados, geração de sinais, cálculos variados, condicionamento de sinais, etc. Além disso, possui numerosos blocos de funções matemáticas como integração, filtros e outros recursos associados à aquisição de dados (WIKPEDIA, 2014).

2.2.3 Aquisição de Dados via Interface Homem-Máquina

Devido aos diversos benefícios descritos na seção 2.2.2, a ferramenta LabVIEW tem sido muito utilizada como uma *interface* homem-máquina para efetuar aquisições de dados. Podendo ainda, proporcionar o desenvolvimento de aplicações variadas como testes de produção e qualidade, verificação de defeitos em produtos e em subsistemas, verificação e validação de projetos, pesquisa e análise, diagnóstico e reparo, monitoramento de condição de ativos, investigação sistemática de projetos, assim como controle e automação de processos sem interação humana baseados em PC (NATIONAL INSTRUMENTS). Devido a todas as características descritas no decorrer da seção 2.2, os autores deste projeto optaram pela ferramenta LabVIEW para realizarem a criação da plataforma didática via *interface* homem-máquina.

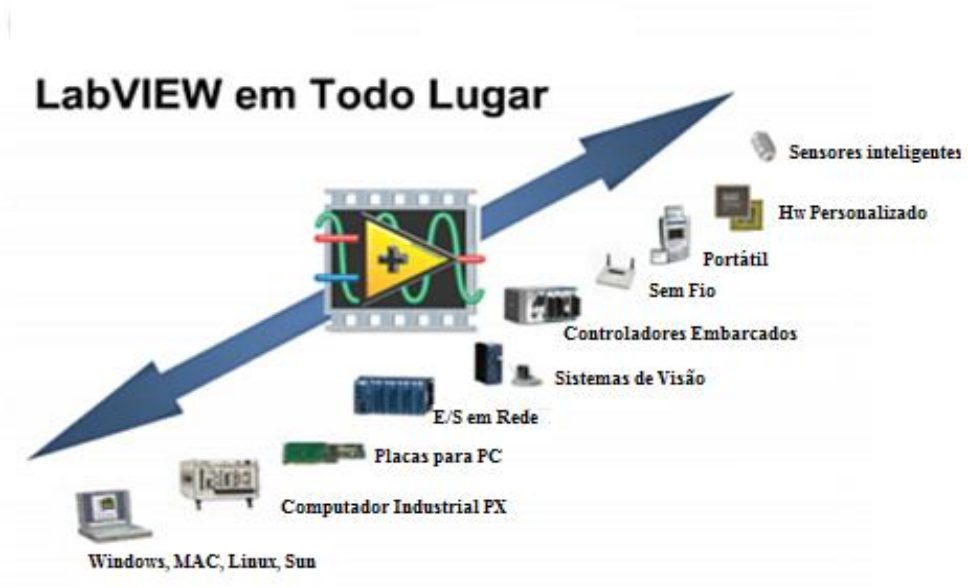


Figura 14 - Algumas aplicações do LabVIEW (Adaptado e disponível em <http://www.rbfautomacao.com.br/o-que-e-labview.html>).

2.3 Condicionamento e Processamento de Sinais

Normalmente, uma aplicação necessita de medições realizadas por intermédio de sinais provenientes de diversos tipos de sensores. No entanto, estes sinais nem sempre estão adequados para serem aplicados diretamente às entradas de um dispositivo de aquisição de dados, sendo necessário um condicionamento dos sinais para que o sistema opere de forma eficaz e exata. Neste capítulo não será abordado profundamente este assunto pelo fato deste projeto se tratar de um kit didático, mas somente será lembrada a importância do mesmo. No caso de implementações futuras por alguém interessado neste aspecto é de extrema importância o estudo e análise dos principais tipos de condicionamento de sinais, tais como amplificação, atenuação, isolação, filtragem, excitação e linearização.

3 Sistemas de Aquisição de Dados

"A invenção é para a tecnologia o que a concepção é para a reprodução: o momento que produz algo original e inédito."

Natan P. Myhrvold

Neste capítulo será apresentado de forma sucinta alguns tipos de sistemas de aquisições de dados, bem como a arquitetura e as características.

3.1 Tipos de Sistemas de Aquisição de Dados

Existem basicamente três tipos de sistemas de aquisição de dados, os sistemas locais, os sistemas remotos e os sistemas *wireless.*

Os sistemas locais são aqueles que estão situados próximos ao elemento que realizará o processamento do sinal. Esta proximidade não deve ultrapassar 30 metros. Ao ultrapassar esta distância, a aplicação se encontra muito distante do sinal processado, passando a ser classificada como aquisição de dados remota, onde o acesso normalmente é realizado por via de cabos ou luz.

Os sistemas *wireless* se enquadram no tipo remoto de sistemas, mas sendo uma alternativa para coletar dados sem a utilização de cabos. Este tipo de comunicação pode ser utilizado em qualquer faixa de frequência livre de licença ou frequências autorizadas (BAPTISTA, 2006).

3.2 Arquitetura de um Sistema de Aquisição Dados

Um sistema de aquisição de dados típico da área de engenharia elétrica possui a capacidade de coletar sinais de sensores e transdutores, assim como condicioná-los para a placa de aquisição de dados, que converte os sinais analógicos para digitais, sendo que um processador interage com a placa de aquisição e com o resto do sistema. Uma forma genérica da arquitetura dos sistemas de aquisições de dados pode ser observada na Figura 15 a seguir:

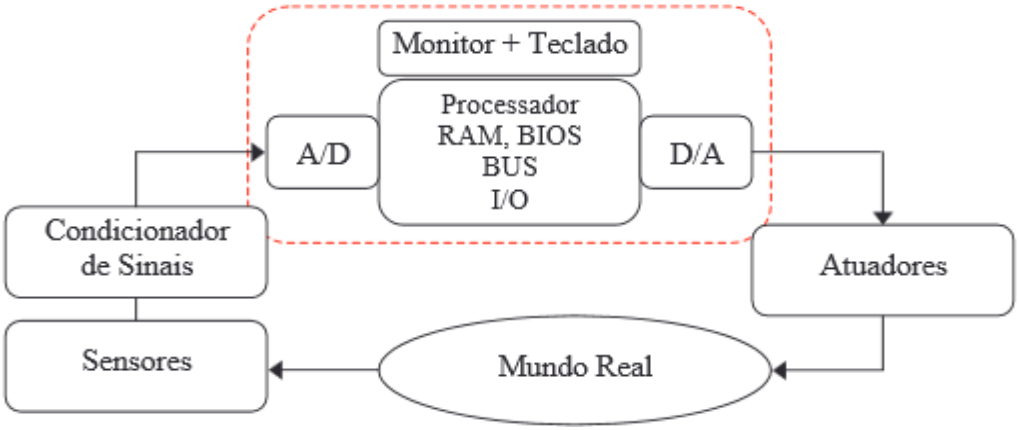


Figura 15 - Arquitetura de um sistema de aquisição de dados [Extraída de (RAMIRES & MURASUG, 2003)].

3.3 Características dos Sistemas de Aquisição de Dados

Um sistema de aquisição de dados normalmente possui certas características específicas como: precisão, resolução, sensibilidade, proteção contra ruído e capacidade de calibração. A precisão dos dados adquiridos vai depender do dispositivo utilizado para efetuar a aquisição e ela não deve ser confundida com a resolução do ADC (*Analog to Digital Converter)*, sendo que a precisão é capaz de mensurar a quantidade de incerteza que existe em uma medição em relação a um padrão relevante e absoluto (RAMIRES & MURASUGI, 2003). Já a resolução descreve o grau pelo qual uma mudança pode ser reconhecida e geralmente é expressa em número de *bits*. Ela é conhecida como "tamanho do degrau" e em um DAC (*Digital to Analog Coverter*), ela é definida como sendo a menor modificação que pode ocorrer em sua saída analógica, resultante de uma alteração na entrada digital. Para relacionar a resolução com valores de medição, devem ser efetuados os devidos cálculos. Por exemplo, em um equipamento com uma resolução de 16 *bits* e uma escala de 20V pico-a-pico, a menor variação que poderia ser detectada por uma medição de tensão seria 305µV (20V / 2^{16}). (BAPTISTA, 2006).

A sensibilidade do sistema é uma quantidade absoluta, ao contrário da resolução que é uma quantidade relativa. A sensibilidade mostra a menor quantidade de alteração no sinal que pode ser detectada pelo sistema de medição.

O ruído nada mais é do que algum sinal indesejável qualquer que se apresenta no sinal que foi digitalizado pelo sistema de aquisição de dados. Por fim, temos a calibração, sendo este um método necessário para projetistas que desejam melhorar a precisão do seu sistema de aquisição de dados. A calibração do *hardware* por intermédio de programas utiliza

conversores digitais-analógicos para realizar compensações, cancelando assim tensões e erros de ganho antes da conversão analógico-digital acontecer. Os dois elementos mais comuns para efetuar calibração de sistemas de aquisições de dados são compensadores e elementos de ganho (BAPTISTA, 2006).

3.4 Sistemas de Aquisição de Dados em Tempo Real

Genericamente, quando um sistema de aquisição de dados consegue acompanhar o estado de um dado processo físico e, se necessário, atuar a tempo sobre ele, então se trata de um sistema de tempo real. Mas é importante ter em mente que tempo real não corresponde à rapidez, mas sim a um ritmo de evolução próprio de um determinado processo físico.

Em relação aos objetivos de um STR (Sistema de Tempo Real) leva-se em conta o tempo de execução (estrutura do código, *pipeline*, sistema operacional), o tempo de resposta (interrupções, acesso a recursos partilhados) e a regularidade de eventos periódicos.

3.4.1 Restrições Temporais

Aplicações de tempo real são caracterizadas por restrições temporais que devem ser respeitadas, a fim de obter o comportamento temporal necessário. As tarefas dessas aplicações tipicamente estão sujeitas a determinados prazos (*deadlines)*, sendo que as tarefas devem ser cumpridas antes de seu *deadline.* Quando uma tarefa é concluída após o seu *deadline* ocorrem consequências que acabam definindo os tipos de restrições temporais, que podem ser classificadas em *soft* (suave), *firm* (firme) e *hard* (rígida) (PEDREIRAS, 2010).

A restrição temporal suave é aquela em que o resultado associado a ela se mantém útil para a aplicação mesmo posteriormente a um determinado *deadline*, embora haja uma queda na qualidade do serviço (o tempo é importante, mas não imprescindível). Um exemplo desta restrição pode ser observado em sistemas bancários, onde há um comprometimento muito maior com a execução das tarefas do que com o tempo de execução das mesmas.

A restrição temporal firme é aquela em que o resultado associado a ela perde toda sua utilidade para a aplicação após um *deadline* específico. Um exemplo desta restrição acontece em videoconferências. Já a restrição temporal rígida, quando não cumprida, é capaz de gerar uma falha catastrófica. Direcionamento de mísseis é um exemplo de STR que exige este tipo de restrição temporal.

4 *Bootloader*

"A construção no presente se dá, não sobre a base de uma repetição e assimilação do passado, mas sobre a construção de diferenças".

Dabdab

Normalmente a maioria dos estudantes utilizam programadores como o PICKit, ICD, PICSTART, PRO MATE, etc. para programarem seus microcontroladores PIC com suporte nativo a USB, pois não têm o conhecimento ou se esquecem de que existe a possibilidade de efetuar a programação dos mesmos sem ter de recorrer a um programador como estes a todo momento que desejarem inserir um novo *firmware* em seus microcontroladores e realizarem seus testes (para a implementação do *Bootloader* só é necessário o uso de um programador uma única vez). Neste capítulo será apresentada a fundamentação teórica básica para a implementação de um *Bootloader* para um microcontrolador PIC da Microchip com suporte a USB.

4.1 Definição de *Bootloader*

Bootloader é um *firmware* que é gravado na memória *flash* de um microcontrolador PIC com suporte a USB. Esse *firmware* é gravado no microcontrolador uma única vez com a utilização de um programador comum, não sendo necessária a utilização deste a partir de então (CEARÁ, 2014). Após a gravação do *Bootloader* é possível gravar um novo *firmware* do programa do usuário através da porta USB, fazendo o uso de aplicativos que são disponibilizados gratuitamente pela Microchip ou outros diversos aplicativos e programas criados por terceiros que adotam essa maneira de gravação. Nas Figuras 16 e 17 pode-se visualizar um exemplo de aplicativo para *Bootloader* da Microchip, sendo mostrado a tela do aplicativo no momento em que não há nenhum dispositivo conectado, a tela do processo de conexão, além da parte de programação do microcontrolador. Estas imagens pertencem ao aplicativo *HID Bootloader*, criado pela Microchip (versão 2.9j), que pode inclusive ser encontrado facilmente em (MICROCHIP ®, 2014).

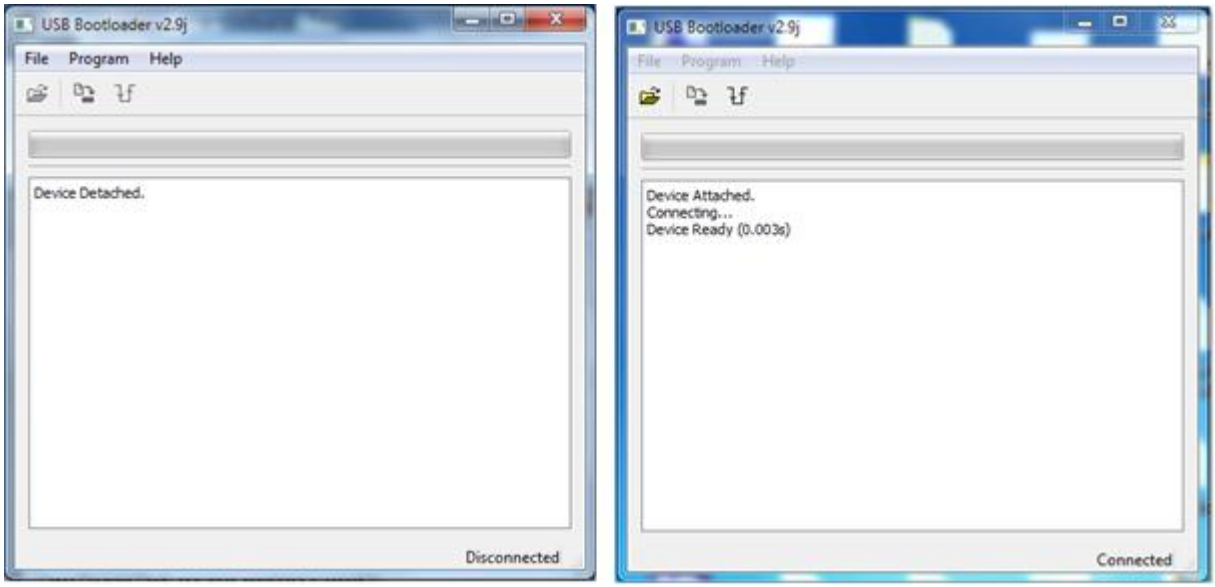


Figura 16 - Tela do aplicativo HID Bootloader: dispositivo desconectado à esquerda e conectado à direita (Fonte: Os autores).

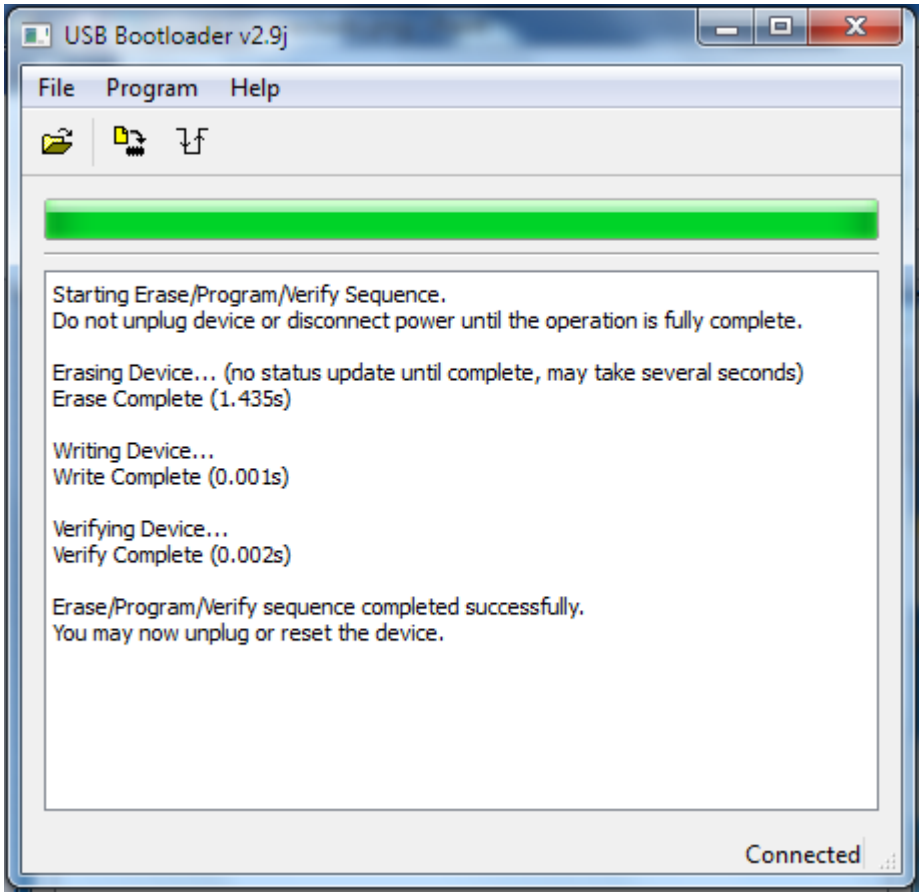


Figura 17 - Tela do aplicativo HID Bootloader: momento de programação do *firmware* (Fonte: Os autores).

4.2 Entrando no modo *Bootloader* via *Firmware*

O *Bootloader* sempre deverá ser o primeiro programa a ser realizado pelo PIC, sendo de extrema importância de que o usuário não se esqueça de "remapear" o início do seu *firmware* principal e os seus respectivos vetores de interrupção e reset para que o *firmware* do *Bootloader* não seja sobrescrito e deixe de funcionar (CEARÁ, 2014). Sendo assim, antes de ser efetuado este "remapeamento" citado anteriormente é necessário que o usuário tenha conhecimento da região de memória que é ocupada pelo *firmware* de seu *Bootloader* (seu início e fim), assim como o seu tamanho em *bytes*.

Na maioria dos *firmwares* de *Bootloader's* é verificada a necessidade de programação de um novo *firmware* (programa de aplicação do usuário) através de uma lógica de programação que verifica o estado de um pino de entrada e saída do PIC atrelado ao estado do pino MCLR (reset) do PIC. Depois que o PIC for *resetado* (pino MCLR em estado baixo), se o estado do pino de entrada e saída for alto, o *Bootloader* fica aguardando para o download do firmware, mas se o estado deste pino for baixo, o *Bootloader* inicia o firmware de aplicação do usuário (STADLER, 2011) e (HENRIQUE, 2013). Com o fluxograma descrito na Figura 18 abaixo é possível entender como é realizada a implementação do modo *Bootloader* via *firmware.*

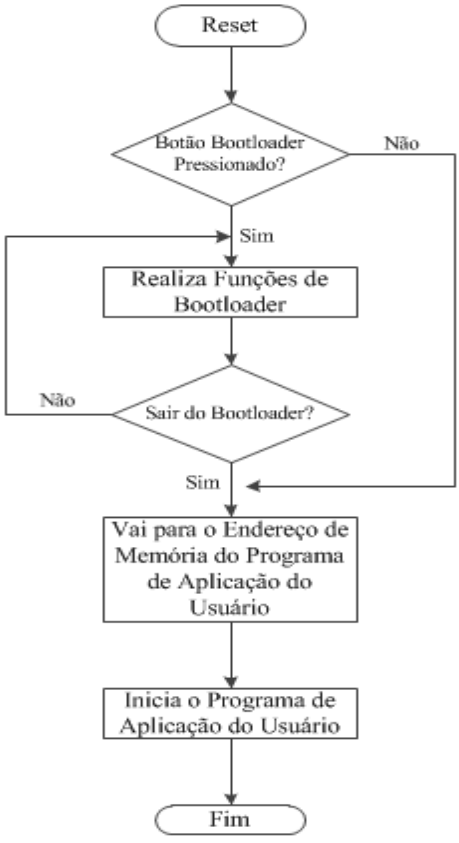


Figura 18 - Fluxograma do modo de entrada do *Bootloader* via *firmware* (Fonte: Os autores).

A Figura 19 permite a visualização de dois mapas de memória para facilitar o entendimento do processo acima referido, sendo um mapa com *Bootloader* e o outro sem *Bootloader*.

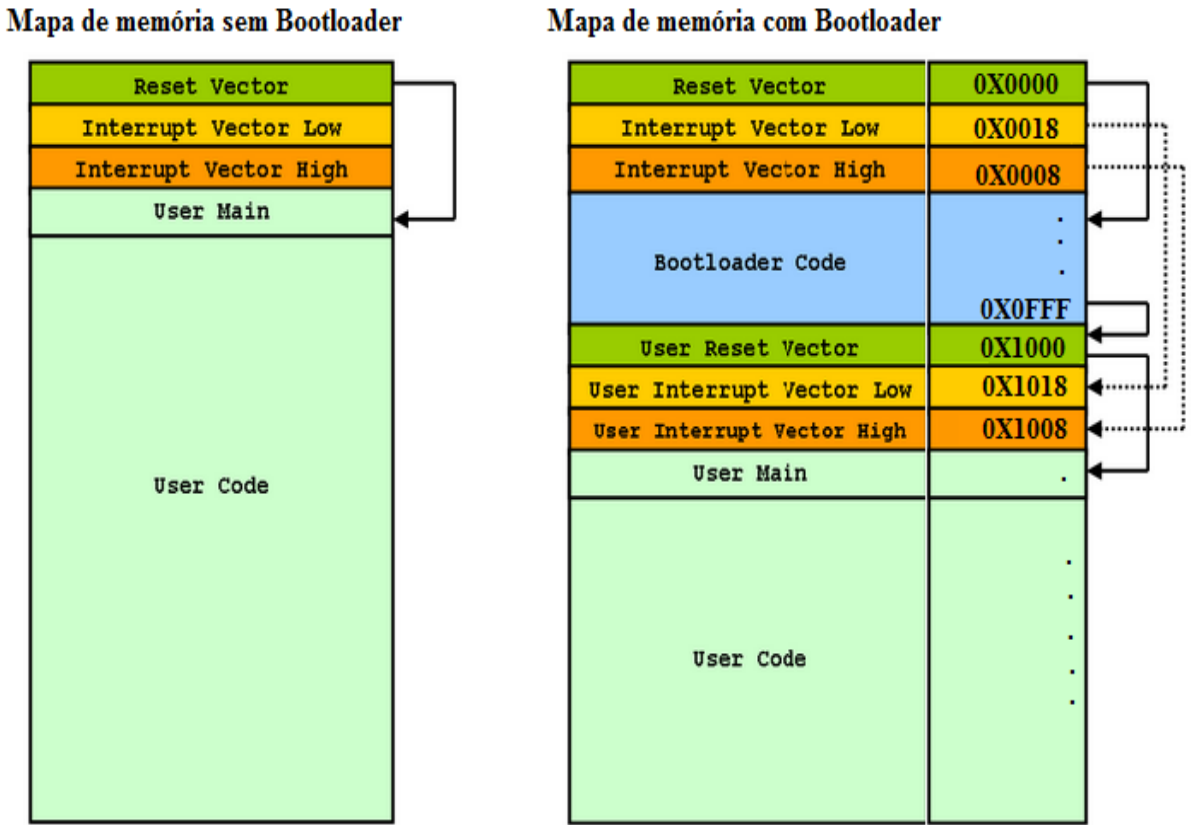


Figura 19 - Mapa de memória com e sem *Bootloader* [Adaptado de (STADLER, 2011)].

As setas que não estão pontilhadas no mapa à direita mostram a sequência de endereçamentos que são seguidos caso o estado do pino de entrada e saída for alto, já as setas pontilhadas mostram a sequência de endereçamentos para o estado baixo deste pino, que faz com que o *Bootloader* inicie o código de aplicação do usuário sem que ocorra a espera para um *download* de *firmware* novo via USB através do aplicativo descrito anteriormente na seção 4.2. Já no mapa à esquerda temos somente o código de aplicação do usuário gravado no PIC, que é iniciado logo após o seu *reset*.

4.3 *Hardware* Necessário

Na Figura 20 é apresentado um circuito básico necessário para a implementação de um *Bootloader* via USB.

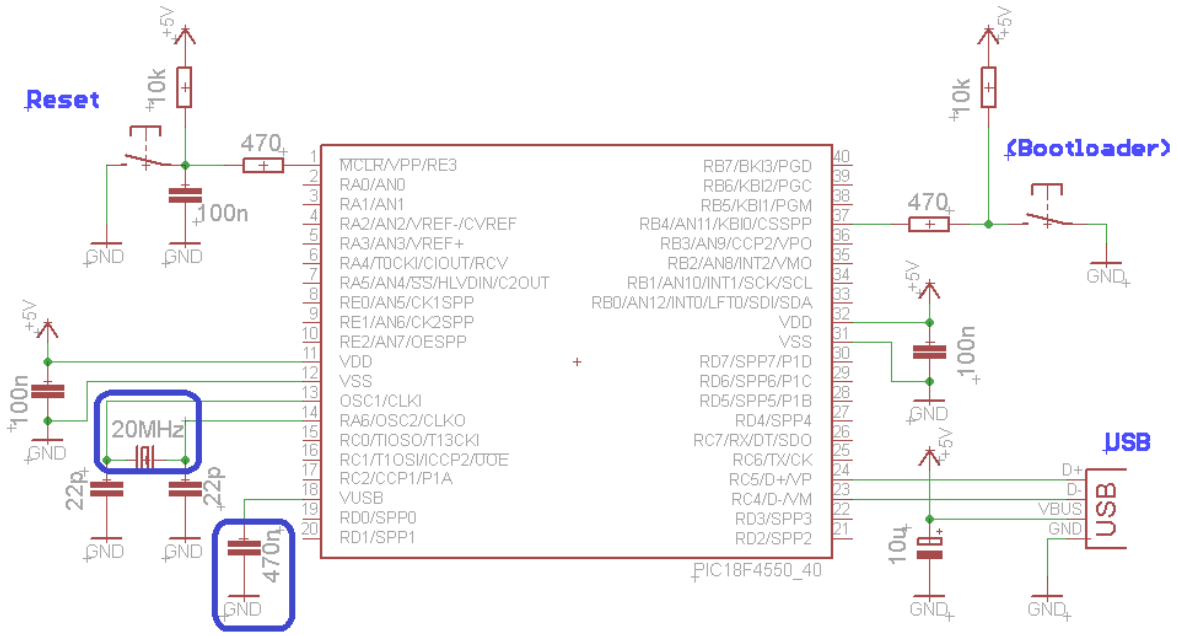


Figura 20 - *Hardware* básico necessário para a implementação do *Bootloader* (Fonte: Os autores).

Um ponto a ser analisado neste circuito é o oscilador de cristal de 20Mhz (mais utilizado normalmente). Destacando que a velocidade em ciclos por segundo requerida para realizar transferência USB é de 48Mhz. Desse modo, devem-se configurar corretamente os geradores internos do PIC (PLL - *Phase-loocked-loop*) para a utilização de outros cristais. Também é importante lembrar que esta propriedade de PLL só está disponível em microcontroladores PIC com suporte a USB (não pode ser encontrada, por exemplo, no PIC 16F877) (ERDOGĂN, 2011).

Outra questão importante é a alimentação do circuito, pois ela é realizada através da porta USB do PC. Se as ligações não forem efetuadas corretamente, podem-se causar danos permanentes ao PC quando conectar o dispositivo na porta USB. O pino Vusb (ligado ao capacitor de 470 nF) do PIC 18F4550 não deve ser confundido com o pino de alimentação do barramento USB.

Na Figura 21 a seguir é apresentado um conector USB tipo A, sendo um para o dispositivo e outro para o PC, ambos com seu diagrama de pinos.

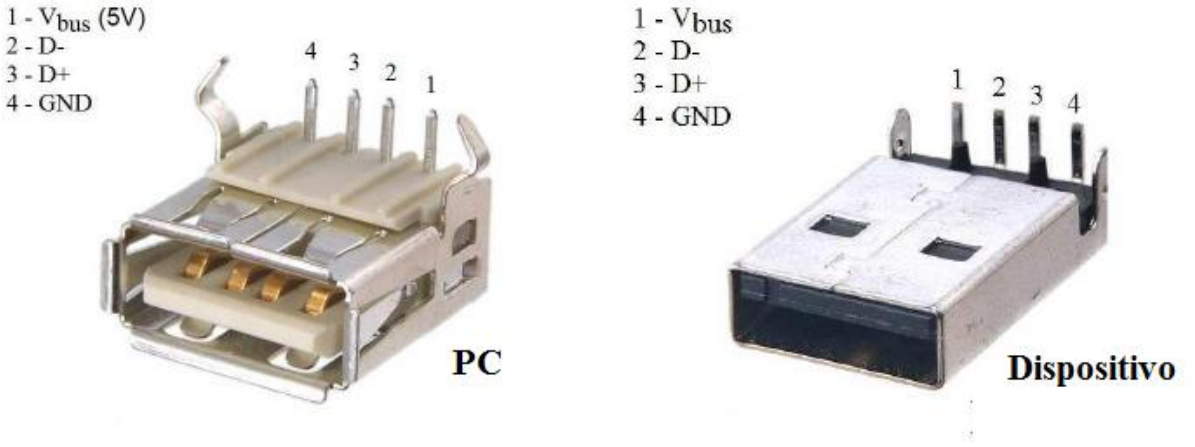


Figura 21 - Conectores USB e suas configurações de pinos [Adaptado de (ERDOGĂN, 2011)].

5 Desenvolvimento do Projeto

"O resumo da sabedoria é este: nunca é perdido o tempo que se consagra ao trabalho."

Ralph Waldo Emerson

Este capítulo mostra descritivamente todo o processo de desenvolvimento desta plataforma didática, que tem como proposta um protótipo de baixo custo com *Bootloader* integrado. Por meio da transferência de dados via USB entre um microcontrolador PIC 18F4550 e a ferramenta computacional LabVIEW, o protótipo desenvolvido é capaz de realizar aquisições e amostrar dados digitais e analógicos, além de poder também gerar certos tipos de sinais por meio do PC através do LabVIEW. Também são abordadas nesta seção as limitações em termos de aquisição de dados.

5.1 Desenvolvimento do *Hardware*

O principal componente de *hardware* explorado neste projeto foi o microcontrolador PIC 18F4550 da Microchip. Este PIC foi escolhido para o desenvolvimento do trabalho pelo fato de possuir muitas funcionalidades, como por exemplo, suporte à USB, além de ser um microcontrolador que é muito utilizado nos projetos da FATEC. Isso acaba por contribuir para o cumprimento de um dos objetivos desta plataforma didática, que é o de auxiliar no avanço e aumento do conhecimento dos alunos que virão a ingressar no curso ou de outras pessoas que vierem a despertar interesse pelo presente trabalho. Na figura 22 pode ser observado o *pin diagram* do microcontrolador referido anteriormente.

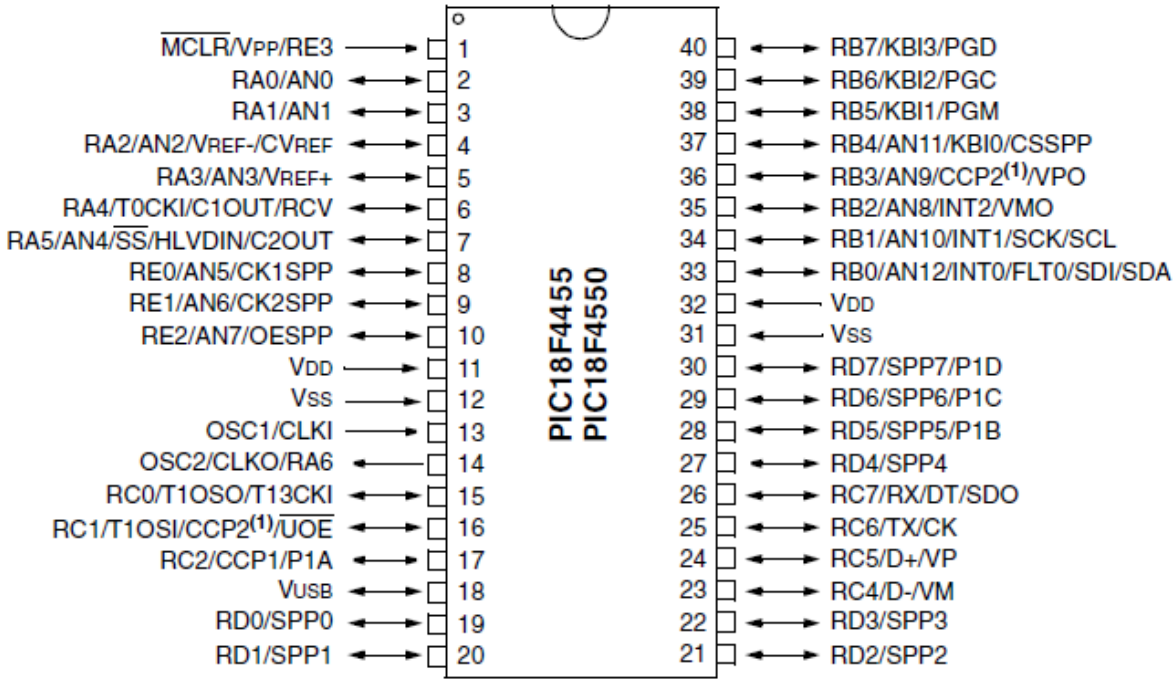


Figura 22 - *Pin diagram* do PIC 18F4550 [Extraído de (MICROCHIP ® *datasheet*, 2006)].

Entre as principais características deste microcontrolador, segundo (SANTOS, 2009) e (SALAS, 2013) podem ser citadas:

- Possui módulo USB 2.0 que suporta *low speed* (1.5Mb/s) e *full speed* (12Mb/s), com 16 *endpoints* bidirecionais. Seu módulo USB suporta os 4 tipos de transferências USB mencionadas na subseção 2.1.4;
- 35 pinos I/O disponíveis;
- Memória *flash* de 32 Kb;
- Memória RAM de 2048 *Bytes*;
- Memória EEPROM de 256 *Bytes*;
- 13 ADC de 10 bits;
- 4 *Timers;*
- Auto-programável por meio de controle interno de *software*, ou seja, o microcontrolador pode escrever na sua própria memória de programação através de uma rotina de *Bootloader*;
- Dois módulos de oscilador externo de até 48Mhz;
- Suporta 100.000 ciclos de apagamento/escrita na memória *flash*;
- Suporta 1.000.000 de ciclos de apagamento/escrita na memória EEPROM;
- Dois módulos CCP;
- Tensão de operação de 4,2V até 5.5V.

A montagem do *hardware* foi realizada conforme a Figura 23. O esquema eletrônico da montagem consta no Anexo 8.1.

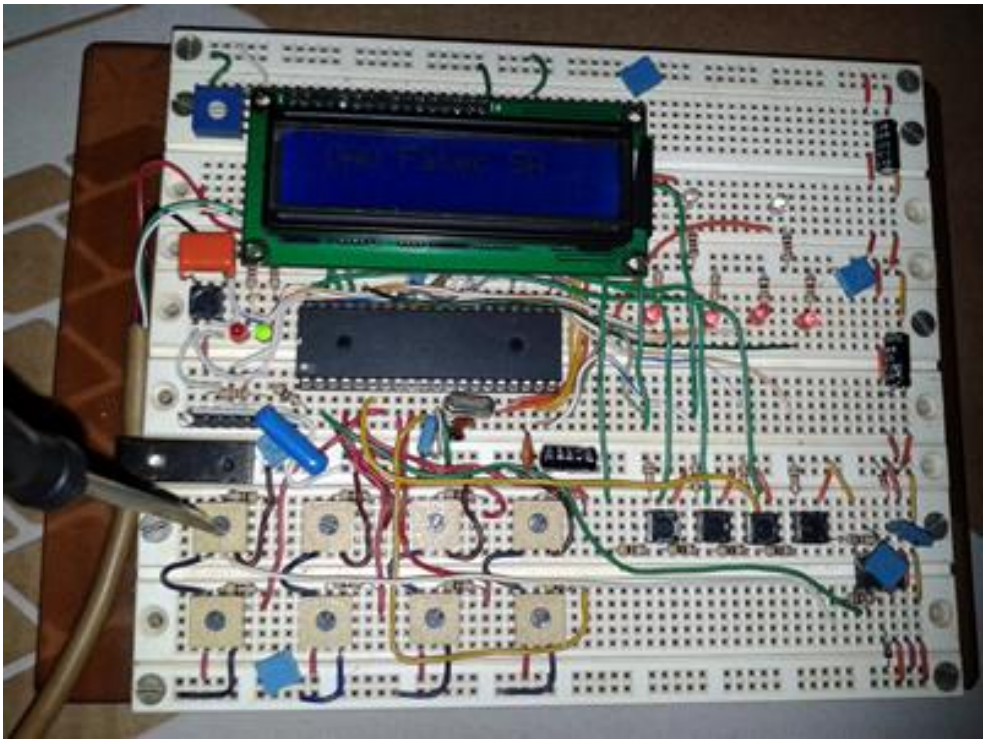


Figura 23 - Montagem do *hardware* em *protoboard* (Fonte: Os autores).

A plataforma didática permite tanto o envio como recebimento de dados, dando certa flexibilidade ao projeto, pois o usuário pode realizar simulações de sensores como também o controle de determinados tipos de atuadores. O envio e recebimento dos pacotes de dados são realizados via USB e o controle e monitoramento é feito através da ferramenta LabVIEW. A Figura 24 ilustra um diagrama em bloco para reforçar a análise do projeto.

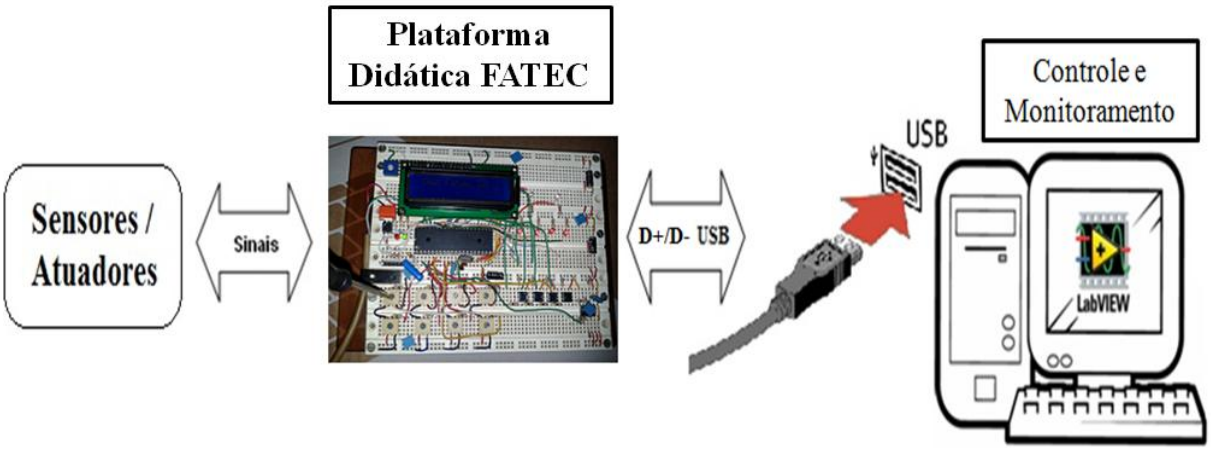


Figura 24 – Diagrama em blocos do projeto (Fonte: Os autores).

5.2 *Firmware* Desenvolvido em Linguagem C no Compliador CCS

A Figura 25 ilustra um fluxograma de nosso *firmware* para melhor entendimento do mesmo.

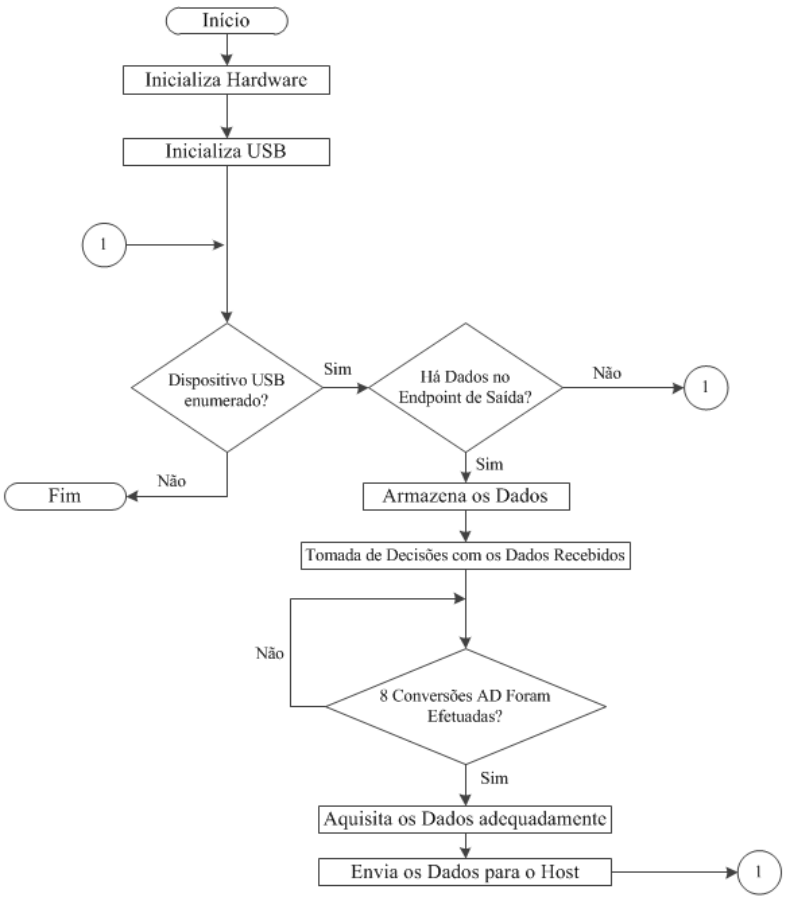


Figura 25 - Fluxograma do *firmware* principal (Fonte: Os autores).

O *firmware* deve conter os descritores para que o *host* USB reúna as informações necessárias para identificar o dispositivo e configurá-lo. Desse modo os descritores possuem a informação básica do dispositivo USB como o número de série, o tipo de transferência suportado, a capacidade de transmissão e outros mais, de modo que o sistema identifique o dispositivo e encontre o *driver* que deve ser utilizado. Para que isso aconteça o *driver* também deve conter os mesmos descritores que o *firmware*.

Os descritores de dispositivo (*device descriptors*) e os descritores de interface (*interface descriptors)* são configurados no *firmware* de acordo com a classe USB do dispositivo, que é definida como *Custom Device Class* e não deve ser confundida com a classe CDC (*Comunication Device Class*), que já foi explicada na seção 2.1.6. Deve-se configurar os descritores para esta classe quando o *driver* mchpusb da Microchip for utilizado. No arquivo do *driver* pode ser visualizado a definição da classe USB:

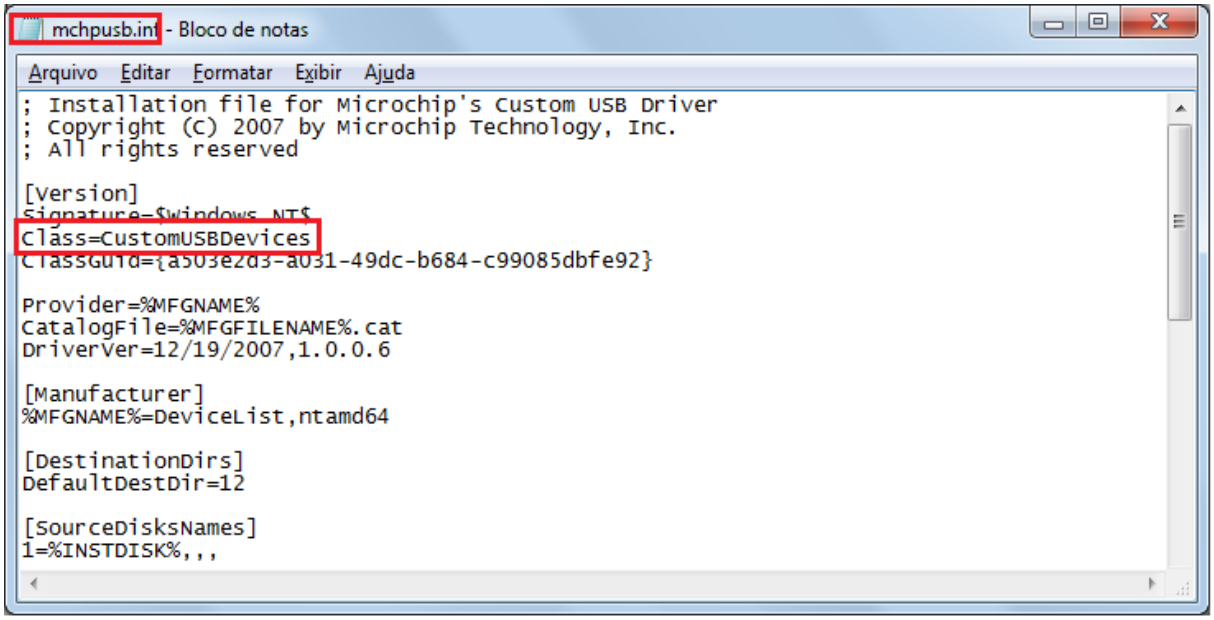


Figura 26 - Definição da classe USB da plataforma didática no *driver* mchpusb (Fonte: Os autores).

Na Figura 27 pode ser observado diversos tipos de códigos de classificação das classes USB:

Base Class	Descriptor Usage	Description
00h	Device	Use class information in the Interface Descriptors
01h	Interface	Audio
02h	Both	Communications and CDC Control
03h	Interface	HID (Human Interface Device)
05h	Interface	Physical
06h	Interface	Image
07h	Interface	Printer
08h	Interface	Mass Storage
09h	Device	Hub
0Ah	Interface	CDC-Data
0Bh	Interface	Smart Card
0Dh	Interface	Content Security
0Eh	Interface	Video
0Fh	Interface	Personal Healthcare
10h	Interface	Audio/Video Devices
11h	Device	Billboard Device Class
DCh	Both	Diagnostic Device
E0h	Interface	Wireless Controller
EFh	Both	Miscellaneous
FEh	Interface	Application Specific
FFh	Both	Vendor Specific

Figura 27 - Códigos de classes USB [Adaptada de (VTM GROUP, 2014)].

Segundo (VTM GROUP, 2014), quando o *device descriptor* é definido com o código 00h, a classe do dispositivo deverá ser definida pela *interface descriptor*. Como a classe *Custom Vendor Specific Class* faz parte da *Custom Device Class*, definimos o nosso *device descriptor* e *interface descriptor* com os códigos 00h e FFh, respectivamente. As definições dos códigos no firmware podem ser observadas na Figura 28.

```
//device descriptor
char const USB_DEVICE_DESC[] ={
      USB_DESC_DEVICE_LEN,
      0x01,
      0x10,0x01,
      0x00, //class code
      0x00, //subclass code
      0x00, //protocol code
        :
        :
//interface descriptor
      USB_DESC_INTERFACE_LEN,
      USB_DESC_INTERFACE_TYPE,
      0x00,
      0x00,
      2,
      0xFF, //class code, FF = vendor defined
      0xFF, //subclass code, FF = vendor
      0xFF, //protocol code, FF = vendor
      0x00,
        :
        :
```

Figura 28 - Fragmento do *firmware* com a definição dos códigos dos *device descriptors* e dos *interface descriptors* (Fonte: Os autores).

Com os descritores configurados corretamente tanto no *driver* como no *firmware*, passamos do processo de enumeração do dispositivo e partimos para a verificação de dados no *endpoint* de saída, ou seja, verificamos se há dados trafegando no *pipe*. No fragmento do *firmware* principal a seguir pode ser observada a metodologia de programação em linguagem C e algumas funções utilizadas para a verificação dos dados no *endpoint* em referência.

```
...

   while (TRUE)
   {
    usb_task();
      if(usb_enumerated())
      {
         if (usb_kbhit(1))
         {
         usb_get_packet(1, dataRX, 3);
         set_pwm1_duty(dataRX[0]);
         set_pwm2_duty(dataRX[1]);
         output_b(dataRX[2]);
         }

...
```

Figura 29 - Fragmento do *firmware* principal responsável por verificar a enumeração do dispositivo USB e se há dados no *endpoint* de saída que foram enviados pelo *host*. (Fonte: Os autores).

A função "usb_task()" é utilizada para habilitar o periférico USB. O primeiro comando de condição "if" é utilizado para verificar se a plataforma didática foi configurada pelo *host*. Se esta condição for verdadeira, verificamos se o *endpoint* de saída contém dados que foram enviados pelo PC através do segundo comando de condição "if" (função "USB_kbhit(1)"). Se esta segunda condição for verdadeira, um pacote de dados com tamanho de 3 *bytes* é armazenado em uma variável vetor do tipo inteiro ("dataRX"), sendo que cada posição deste vetor possui 1 *byte* de tamanho. Desse modo podemos armazenar nas posições do vetor um valor de 0 a 255, valor este que é enviado pelo *host* via LabVIEW e utilizado para "setar" o valor do *duty cicle* do sinal PWM que é enviado para dois LEDs em nossa plataforma didática. O terceiro *byte* é utilizado para armazenar dados enviados pelo LabVIEW (comando para os LEDs ligados no portB do PIC).

Depois de realizado as tomadas de decisões com os dados recebidos, partimos para a lógica de programação responsável por fazer com que um pacote de dados de 64 *bytes* seja enviado do PIC para o *host*. Trata-se de um laço de iteração "for", em que cada uma das 8 iterações do laço é responsável por selecionar um conversor AD do PIC18F4550. Em nosso *firmware* optamos por enviar somente 10 dos 64 *bytes* do pacote, sendo estes armazenados em um variável vetor do tipo inteira que denominamos como "dataTX". Os dois últimos *bytes* (posição 8 e 9 do vetor) são utilizados para enviar o retorno do valor da contagem do Timer0 e o estado lógico de *push-bottoms* ligados ao portB do PIC. Tudo o que foi acima descrito pode ser observado na Figura 30.

```
...
            for(count=0;count<8;count++)
            {
            set_adc_channel(count);
            delay_us(20);
            dataTX[count]=read_adc();
            }
            dataTX[8]=get_timer0();
            dataTX[9]=input_b();
            usb_put_packet(1, dataTX,64, USB_DTS_TOGGLE);
        }
    }
}

...
```

Figura 30 - Fragmento do *firmware* principal responsável por enviar dados do PIC para o *host*. (Fonte: Os autores).

O *firmware* completo pode ser encontrado no Anexo 8.3.

5.3 A Programação em Linguagem G pelo LabVIEW

A interface LabVIEW foi desenvolvida em duas partes, composta por uma VI principal e por uma subVI, que foram nomeadas como “Interface_BR” e “Interface_BR_sub”, consecutivamente. A subVI é estruturada com a finalidade de manipular corretamente a DLL (*Dynamic Link Library*) mpusbapi da Microchip e a VI principal tem como responsabilidade a manipulação dos pacotes de dados que são enviados e/ou recebidos pelo *host*. Mais detalhes poderão ser vistos nos subcapítulos 5.3.1 e 5.3.2 adiante.

5.3.1 SubVI "Interface_BR_sub" em Detalhes

Após o desenvolvimento das etapas descritas nos capítulos 5.1 e 5.2 foi realizado a programação gráfica do projeto em LabVIEW para que fosse efetuada a transferência de dados entre o microcontrolador PIC 18F4550 e o *host*, sendo este no projeto um *notebook* comum com portas USB 2.0.

Para estabelecer a comunicação entre o *host* e o microcontrolador foi necessário utilizar uma função do LabVIEW chamada *call library function node*. Esta função foi explorada na subVI nomeada como "Interface_BR_sub" e pode ser visualizada na Figura 31 abaixo.

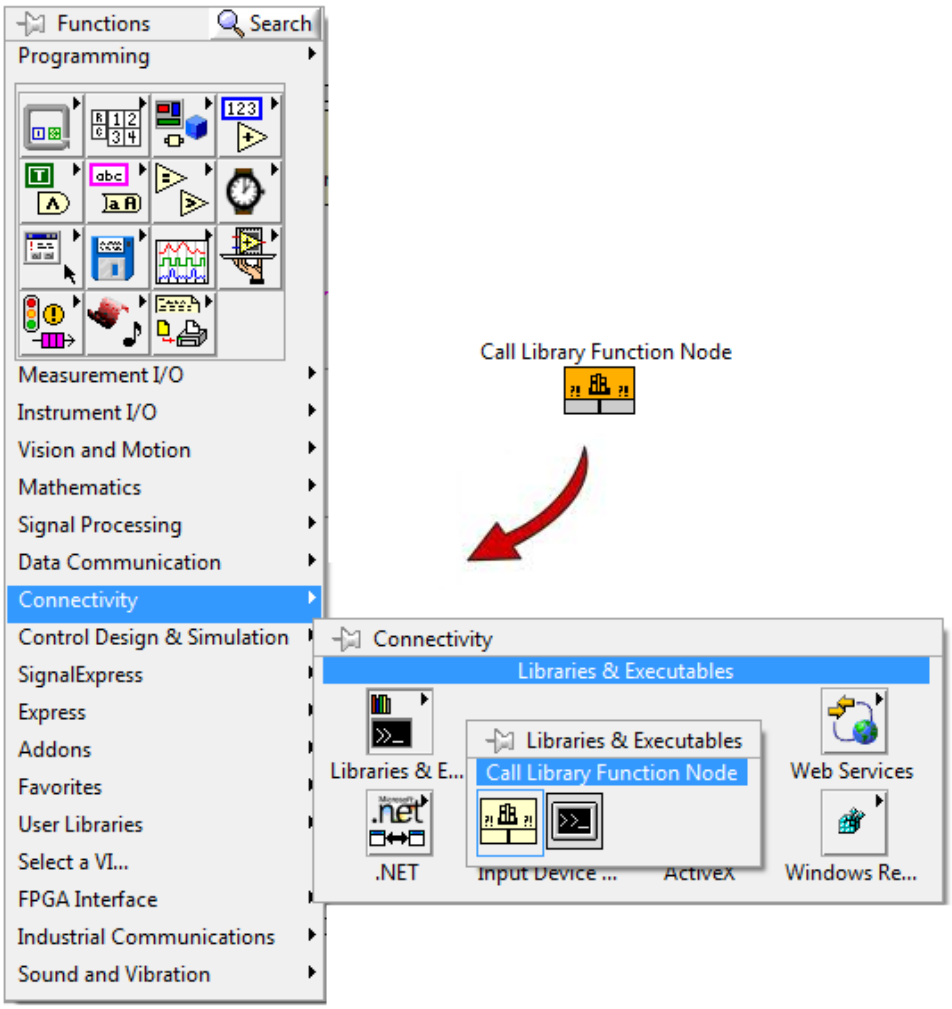


Figura 31 - Função *call library node* do LabVIEW (Fonte: Os autores).

Através dessa função conseguimos fazer o uso adequado da DLL mpusbapi que é disponibilizada pela Microchip. Esta DLL contém um código encapsulante com funções simples que podem ser usadas para acessar *endpoints* de transferências USB por interrupção,

do tipo *bulk* ou isossíncrona, simplificando assim o desenvolvimento de aplicações com o *driver* mchpusb, desenvolvido pela Microchip, que será detalhado no capítulo 5.4. Além disso a DLL é gratuita.

Neste projeto foram utilizadas algumas funções básicas da DLL mpusbapi por meio da função ilustrada na Figura 31, sendo que a primeira a ser utilizada foi a função MPUSBGetDeviceCount, conforme ilustra a Figura 32 a seguir, que permite a visualização da primeira sequência do *sequence flat* da subVI do projeto.

Neste primeiro passo ilustrado na Figura 32 verificamos a quantidade de dispositivos conectados com o VID e PID fornecido (valor de retorno da função).

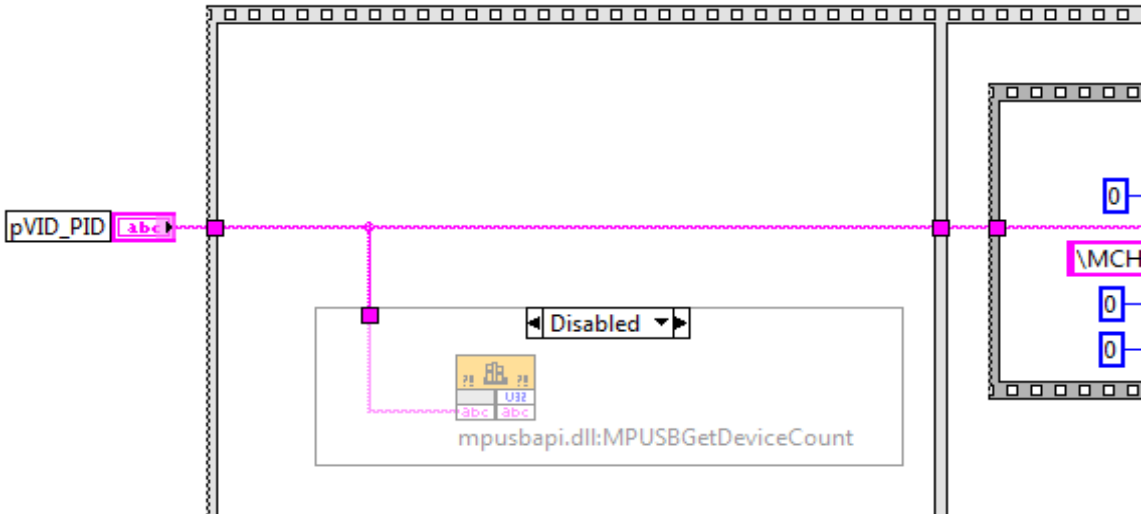


Figura 32 - Utilização da função MPUSBGetDeviceCount da DLL mpusbapi (Fonte: Os autores).

O segundo passo da *sequence flat* pode ser verificado na Figura 33 a seguir, onde é utilizada a função MPUSBOpen da DLL mpusbapi. O retorno desta função é necessário para a abertura dos *pipes* de comunicação (*pipe in* e *pipe out*) do dispositivo USB com o PID e VID especificado. O parâmetro de entrada dwDIR especifica a direção do *endpoint* (entrada e saída ou leitura e escrita). O parâmetro de entrada *Instance* especifica um número de dispositivo para abrir, mas como efetuamos primeiramente a chamada da função MPUSBGetDeviceCount, atribuímos o valor zero para este parâmetro. Já o parâmetro de entrada pEP é uma *string* que contém o número do *endpoint* que será utilizado, sendo que esta *string* deve ser escrita na forma "\MCHP_EPx", onde x é o número do *endpoint* (neste projeto utilizamos o *endpoint* 1). O parâmetro de entrada dwReserved indica apenas um número reservado para o dispositivo USB, não sendo utilizado neste projeto. Por fim, temos o parâmetro de saída *handle*, que é o retorno da função e será utilizado nas outras sequências do *sequence flat*. Outra observação a ser feita na Figura 33 é que ilustramos as duas sequências

do *sequence stack* na mesma imagem para visualização das configurações do *pipe* de entrada e do *pipe* de saída.

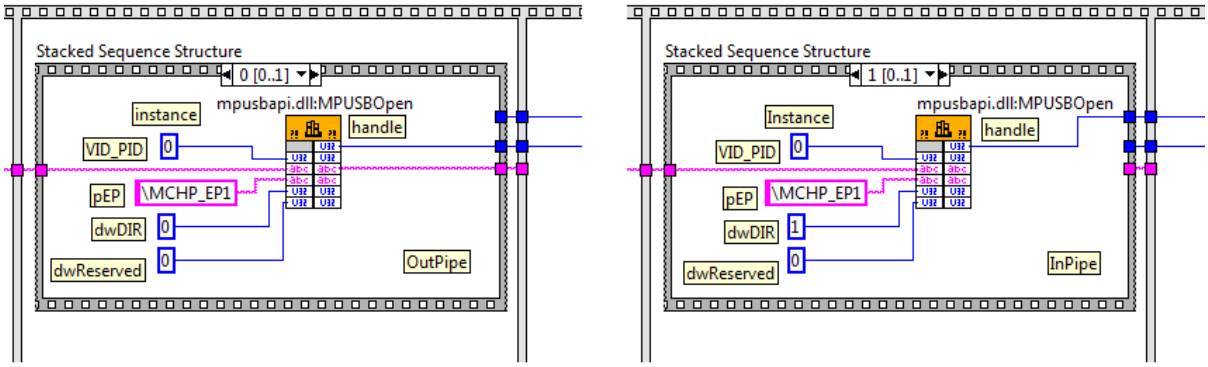


Figura 33 - Utilização da função MPUSBOpen da DLL mpusbapi (Fonte: Os autores).

O nosso terceiro passo da *sequence flat* consiste no uso da função MPUSBWrite da DLL mpusbapi, que é utilizada para a escrita de dados e poderá ser visualizada na Figura 34 logo abaixo deste parágrafo. O parâmetro de entrada handle identifica o *pipe* do *endpoint* que se deseja escrever (no nosso caso este parâmetro é o próprio *handle* de saída do primeiro *stack* da Figura 32). Já o parâmetro de entrada pData nós renomeamos para "Dados a enviar", que é um ponteiro para o *buffer* que contém os dados que serão enviados. Outro parâmetro de entrada, o dwLen, também foi renomeado no projeto, passando a ser denominado como "Bytes a Enviar". Este parâmetro é responsável por especificar o número de *bytes* a serem escritos no *pipe*. Também renomeamos o parâmetro de entrada dwMilliseconds para "TimeOut Escrita", que nada mais é do que um período para realizar uma nova tentativa, um *retry*. Por último, temos o parâmetro de saída pLenght, que representa o número de *bytes* que efetivamente foram escritos, renomeado neste projeto como "Bytes Enviados".

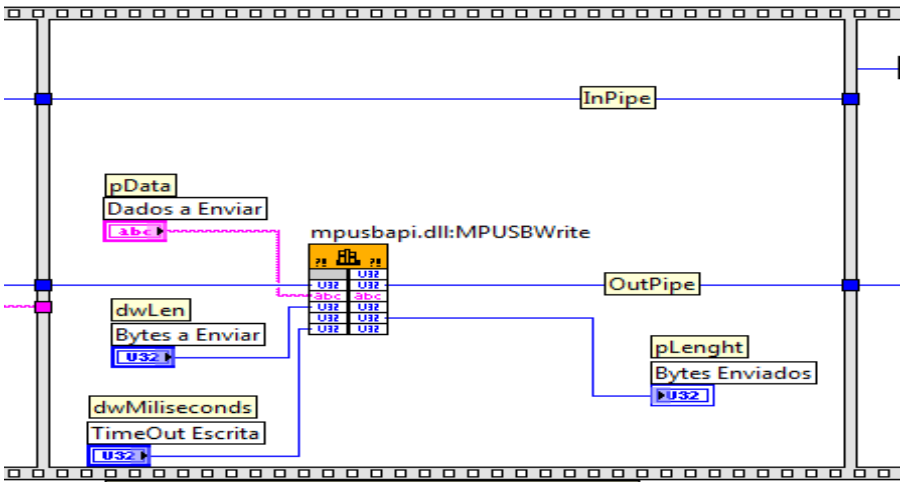


Figura 34 - Utilização da função MPUSBWrite da DLL mpusbapi (Fonte: Os autores).

Em relação ao nosso quarto passo da *sequence flat* desta subVI, exploramos a função MPUSBRead da DLL mpusbapi, que possui como parâmetros de entrada o handle, dwLen e dwMilliseconds, renomeados consecutivamente como “InPipe”, “Bytes a Ler” e “TimeOut Leitura”. O *handle* identifica o *pipe* do *endpoint* que se deseja ler, o dwLen especifica o número de *bytes* que serão lidos no *pipe* e o dwMilliseconds continua com a função de *retry* explicada anteriormente. Por fim, temos os parâmetros de saída pData e pLenght, renomeados consecutivamente para “Dados Recebidos” e “Bytes Lidos”. O parâmetro pData é um ponteiro para o *buffer* que contém os dados lidos do *pipe* e o parâmetro pLenght que representa o número de bytes que efetivamente foram lidos. Quando “Bytes Lidos” for diferente de zero, o nível lógico na saída do comparador se torna verdadeiro, sendo que utilizamos este nível lógico para verificar se o dispositivo USB foi encontrado. Tudo isso pode ser visualizado na Figura 35 a seguir.

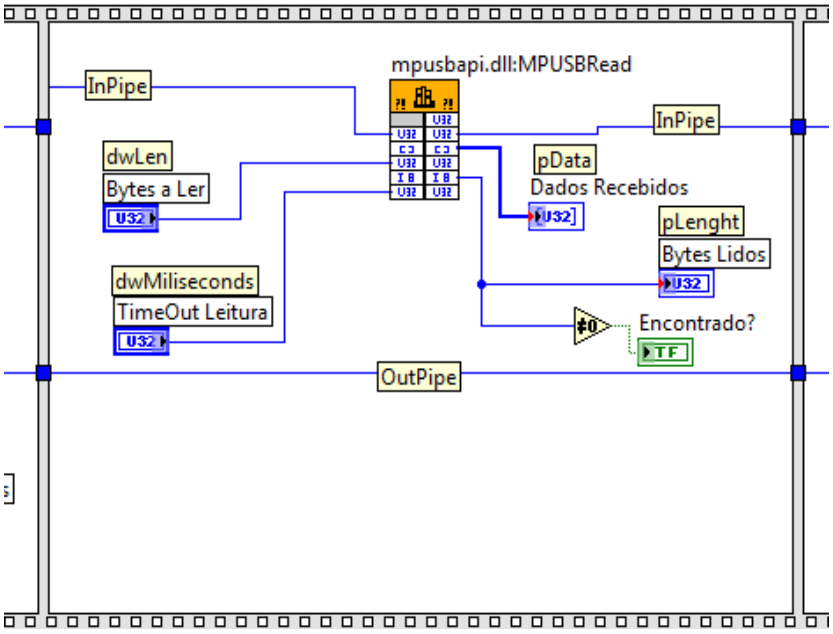


Figura 35 - Utilização da função MPUSBRead da DLL mpusbapi (Fonte: Os autores).

O quinto e sexto passo consiste na utilização da função MPUSBClose da DLL mpusbapi, que possui somente o parâmetro de entrada *handle*, com a função de identificar o *pipe* do *endpoint* que se deseja fechar. Neste projeto utilizamos duas vezes essa função com a finalidade de encerrar o *pipe* de entrada e o *pipe* de saída. Tudo isso pode ser observado na Figura 36 logo abaixo.

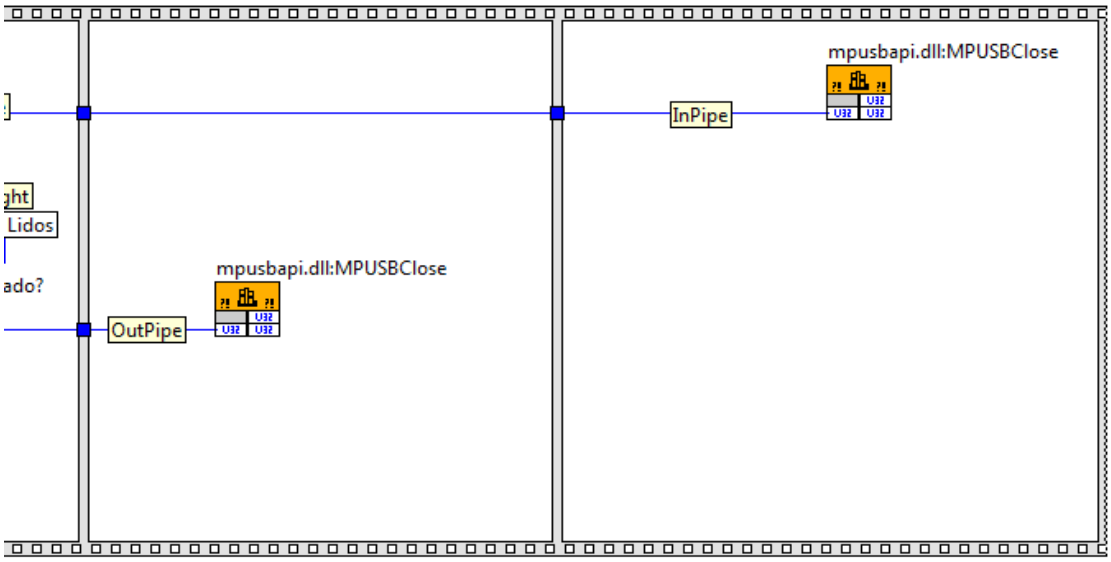


Figura 36 - Utilização da função MPUSBClose da DLL mpusbapi (Fonte: Os autores).

Uma observação importante que deve ser feita é que nessa subVI há diversos parâmetros que estão como controles e indicadores, pois eles são somente terminais de ligação da subVI. Os valores adotados para tais parâmetros serão encontrados na VI principal do projeto, nomeada como Datalogger_BR, conforme será detalhado na seção 5.3.2.

Abaixo segue uma ilustração do painel frontal da subVI "Interface_BR_sub":

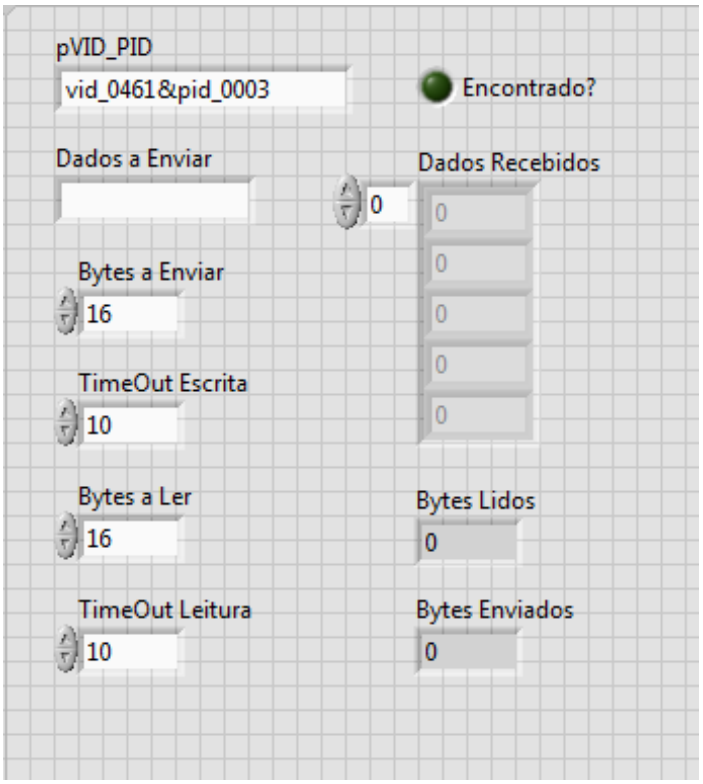


Figura 37 - Painel frontal da subVI do projeto (Fonte: Os autores).

Por fim, apresentamos um fluxograma para melhor compreensão desta subVI na Figura 38.

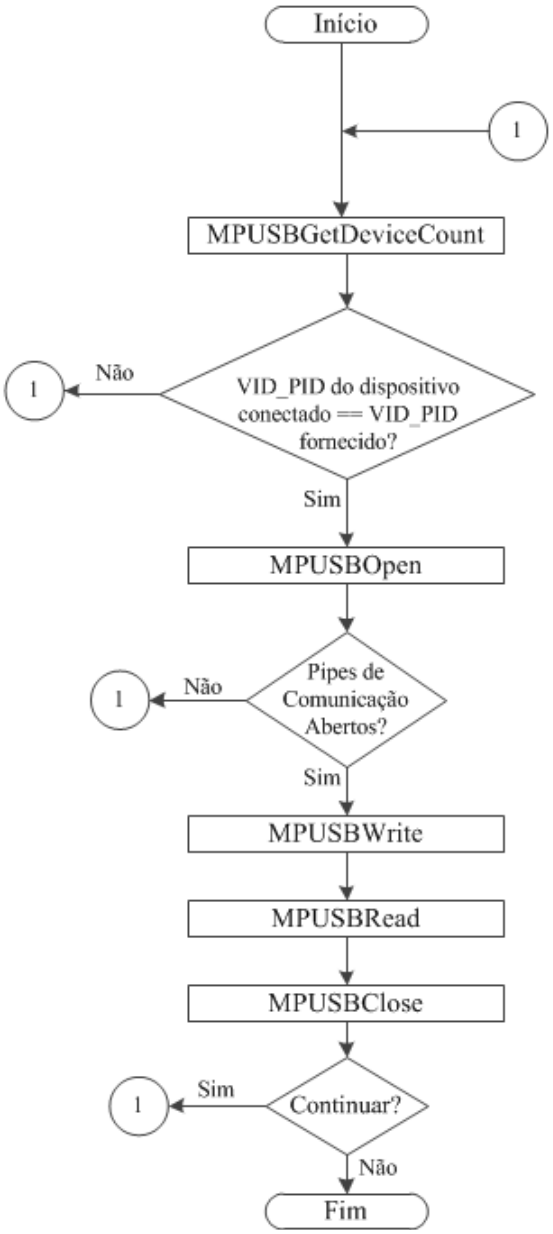


Figura 38 - Fluxograma da subVI Interface_BR_sub (Fonte: Os autores).

5.3.2 Desenvolvimento da VI Principal

Nesta seção detalharemos o desenvolvimento da programação em LabVIEW da nossa VI principal do projeto em sua fase inicial, nomeada como "Interface_BR" (os detalhes finais podem ser observados na seção 5.5). De início é importante analisar as imagens do painel frontal e do diagrama de bloco correspondente a esta VI para que seja possível a compreensão desta etapa do projeto. Estas ilustrações podem ser visualizadas abaixo nas Figuras 39 e 40:

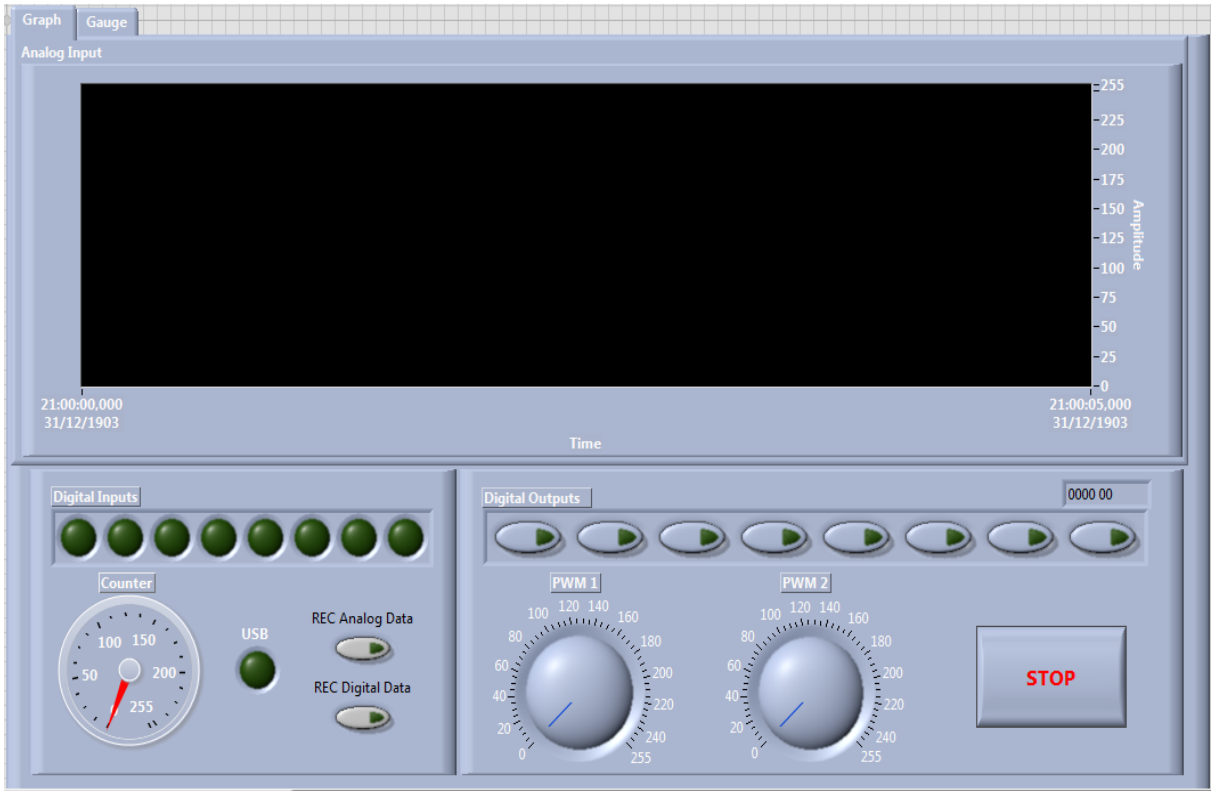


Figura 39 - Painel frontal da VI principal do projeto (Fonte: Os autores).

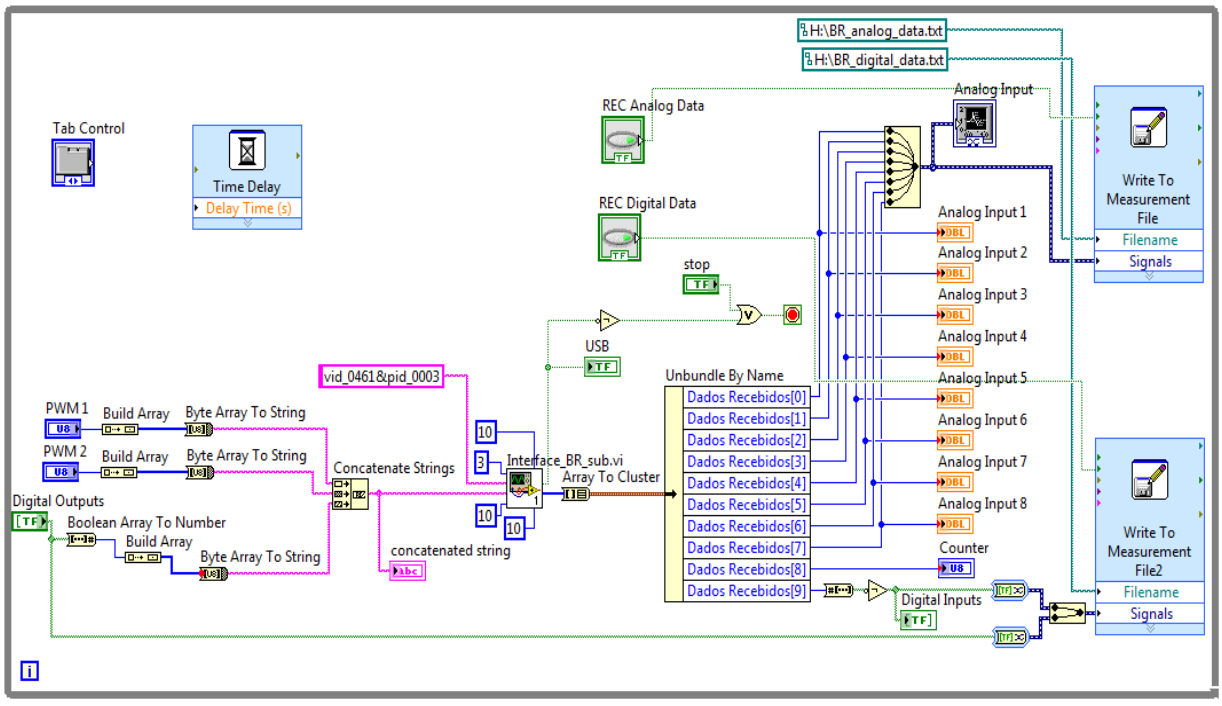


Figura 40 - Diagrama de bloco da VI principal do projeto (Fonte: Os autores).

A VI será detalhada por partes para facilitar o entendimento, sendo também associada com o *firmware* principal do projeto. É importante lembrar que o fluxo de dados desta VI é estruturado por meio de um laço *while*, sendo que as iterações do laço ocorrem a cada 1 milissegundo (tempo que atribuímos na função "Time Delay" do LabVIEW). Este *delay* ajuda na visualização dos dados coletados e que são amostrados no "waveform graph" do LabVIEW.

Na figura 41 a seguir podemos visualizar como iremos enviar os dados através do *host* para o *hardware* da plataforma didática por meio da transferência USB.

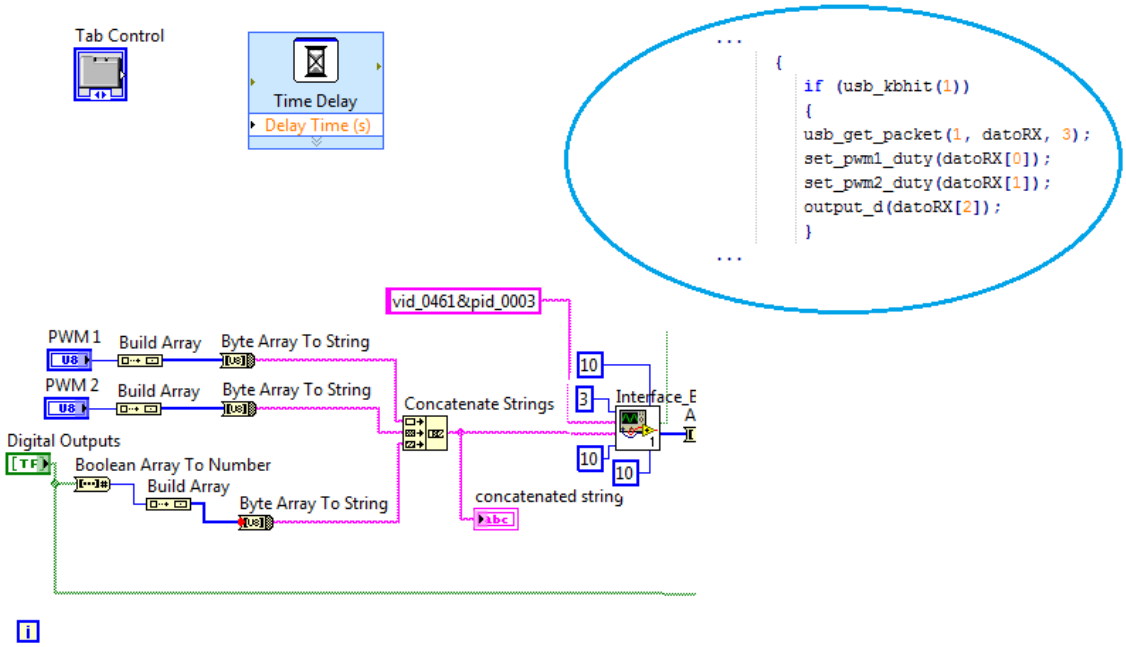


Figura 41 - Seção do *Block Diagram* responsável pela transferência de dados do *host* para a plataforma didática e suas correlações com o *firmware* (Fonte: Os autores).

Os pacotes de dados que são enviados pelo *host* são compostos por 3 *bytes*, sendo o primeiro e segundo *byte* responsável pela informação do *duty cicle* do sinal PWM (*Pulse Width Modulation*), conforme ilustra o fragmento do *firmware* da Figura 41 (como cada *byte* possui 8 bits, o *duty cicle* pode variar de 0 a 255). O terceiro *byte* tem a função de enviar 8 sinais digitais para o Portd do PIC 18F4550. Estes dados são armazenados em um variável vetor dentro do *firmware* principal, conforme foi detalhado na seção 5.2 e também como ilustrado na parte superior direita da figura 41. Para que a transferência ocorra de forma correta deve-se estabelecer uma coerência na ordem dos *bytes* contidos no pacote de dados, sendo que a ordem no diagrama de blocos do LabVIEW não deve diferenciar-se da ordem do variável vetor do *firmware*. Desse modo, como o parâmetro de entrada da subVI responsável pelo envio de dados para o PIC é do tipo *string*, (parâmetro "Dados a Enviar"), o único método que encontramos para ordenar o pacote de envio de dados foi adaptar os *bytes*

enviados para o tipo *string* e concatenar os mesmos através da função “concatenate strings” do LabVIEW, conforme ilustrado na figura 41.

A figura 42 abaixo mostra a parte do painel frontal responsável pelo controle dos dados que serão enviados do *host* para a plataforma didática.

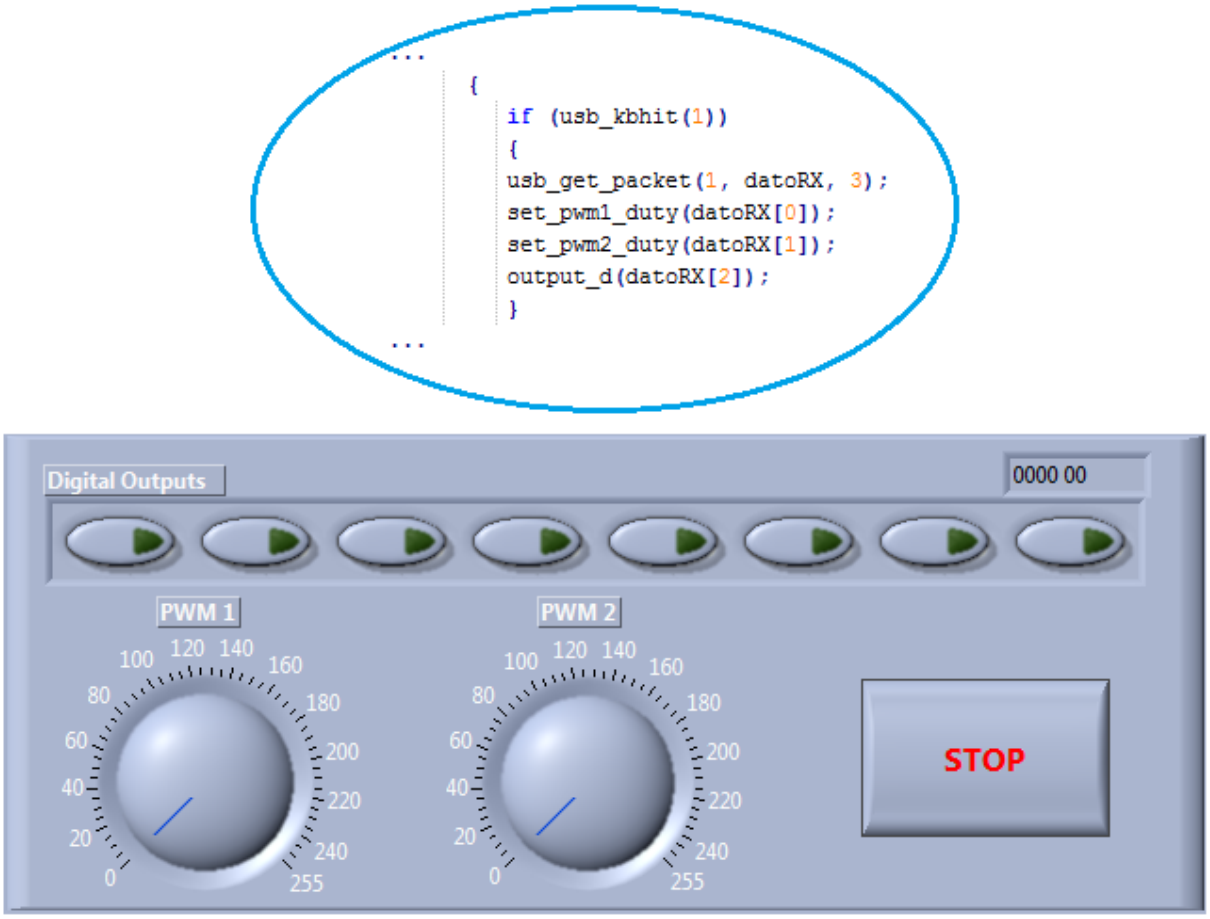


Figura 42 - Seção do *Front Panel* responsável pela transferência de dados do *host* para a plataforma didática e suas correlações com o *firmware* (Fonte: Os autores).

Seguindo o fluxo de dados da VI principal, temos os valores que adotamos para as entradas da subVI, sendo o valor 3 atribuído ao parâmetro “Bytes a Enviar” e o valor 10 atribuído aos parâmetros “Bytes a Ler”, “TimeOut Escrita” e “TimeOut Leitura”. Outro valor muito importante é o VID e PID, sendo importante lembrar novamente da coerência com os valores contidos no *firmware*. Mais detalhes sobre o VID e PID poderão ser observados na seção 5.4, onde iremos descrever o processo de detecção e enumeração USB de nossa plataforma didática. Temos também detalhes da programação gráfica utilizada neste projeto para realizar a visualização da detecção de nossa plataforma didática USB. Toda esta continuidade do fluxo de dados citada neste parágrafo pode ser observada na figura 43.

A lógica para a visualização da detecção USB da plataforma didática no LabVIEW foi realizada a partir do parâmetro de saída da subVI “Encontrado?”, sendo este valor verdadeiro somente quando há alguma leitura de dados no *pipe* de entrada (*in pipe*). E foi por meio deste parâmetro que também desenvolvemos a lógica de programação gráfica para bloquear o laço *while* caso não ocorra a enumeração USB de nosso dispositivo. Este bloqueio é feito através da saída de uma porta lógica OR (se o valor de “Encontrado?” for falso, a porta lógica NOT inverte seu estado, realizando o bloqueio do laço *while* e consequentemente não ocorre a execução do programa). Toda esta lógica de programação gráfica citada neste parágrafo também pode ser observada na figura 43.

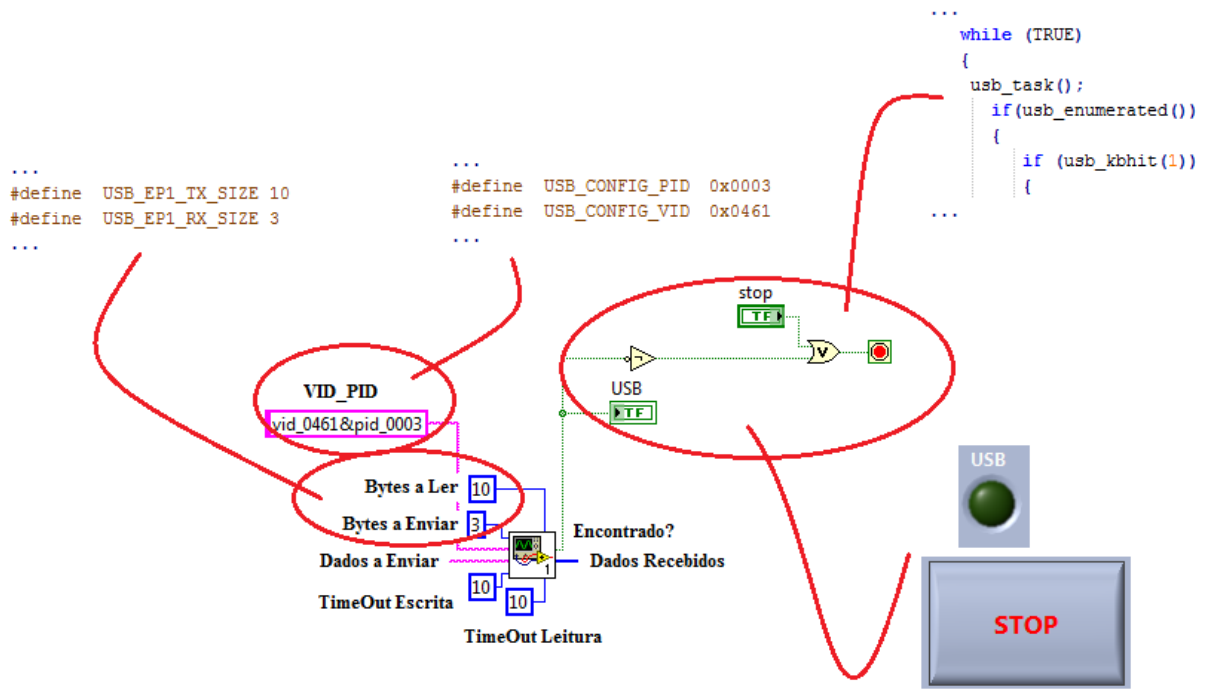


Figura 43 - Parâmetros da subVI do projeto e suas correlações com o *firmware* (Fonte: Os autores).

Se por um lado realizamos o envio de dados para o *host* pelo *pipe* de saída no LabVIEW (*out pipe*), também recebemos dados enviados pela nossa plataforma didática para o *host* através do *pipe* de entrada no LabVIEW (*in pipe*). Nossas configurações de *firmware* da plataforma didática em conjunto com os valores adotados no *Block Diagram* do LabVIEW permitem que seja enviado do PIC para o *host* pacotes de dados compostos por 10 *bytes*, mas vale ressaltar que com o tipo de comunicação USB utilizada neste trabalho (transferência USB do tipo *bulk*), em conjunto com as bibliotecas USB utilizadas, é possível o envio de pacotes de dados de no máximo 64 *bytes* (no final do projeto optamos por fazer o envio de um pacote 64 *bytes*, mas fazendo o uso somente dos 10 *bytes* citados anteriormente).

Como o LabVIEW recebe do PIC 18F4550 de nossa plataforma um pacote de dados de 10 *bytes,* resolvemos inserir este pacote em um *cluster* no LabVIEW para trabalhar de maneira adequada com cada um dos *bytes* contidos neste pacote. A figura 44 ilustra claramente o tratamento que utilizamos com cada um dos 10 *bytes* que são enviados em pacotes por nosso dispositivo USB, assim como a correlação com a programação de nosso *firmware*.

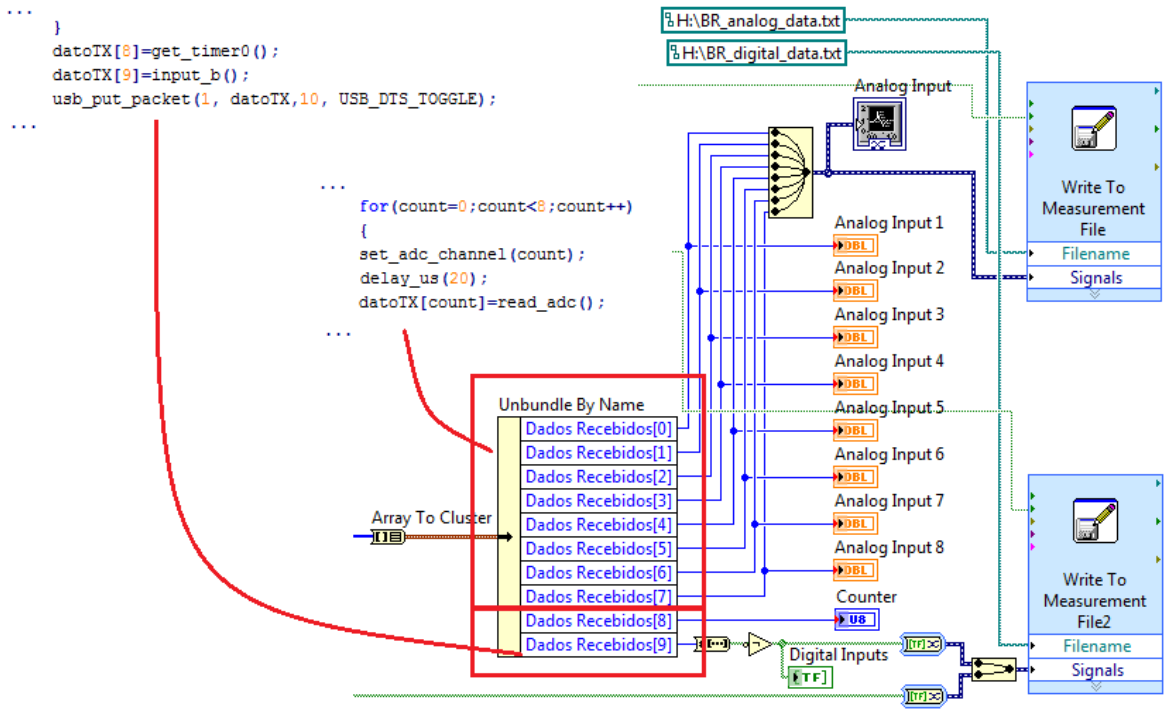


Figura 44 - Seção do *Block Diagram* responsável pela transferência de dados da plataforma didática para o *host* e suas correlações com o *firmware* (Fonte: Os autores).

Utilizamos os 8 primeiros *bytes* do pacote para coletar os sinais analógicos gerados por 8 potenciômetros ligados em 8 entradas analógicas do PIC 18F4550 conforme anexo 8.1. A aquisição é feita por meio do ADC do PIC, fazendo o uso de todos os 8 canais disponíveis. Como o pacote USB é formado por *bytes*, optamos pela resolução de 8 bits do ADC, fazendo com que os dados analógicos de entrada variem de 0 à 255 no *display* do "waveform graph" presente no *Front Panel* de nossa VI principal. Em relação ao *byte* 9 optamos pela inclusão de um simples contador em nossa plataforma. Isto foi contemplado com a utilização do Timer0 do PIC 18F4550. Como o valor representado pelo Timer0 é do tipo inteiro, no *Front Panel* da VI principal poderá ser visualizado a contagem realizada peloTimer0, valor este que também variará de 0 à 255.Por fim temos o *byte* 10 do pacote, sendo este responsável por armazenar o estado lógico de 8 *push-buttons* conectados ao Portb de nosso PIC 18F4550. Dessa maneira, ao pressionarmos fisicamente qualquer um desses *push-bottons* da plataforma didática, haverá

comutações dos estados digitais dos LEDs presentes no *Front Panel* da VI principal (nomeados com "Digital Inputs").

Na Figura 44, também é possível perceber a metodologia que utilizamos para realizar a gravação dos dados captados. Para esta tarefa utilizamos duas funções "*write to measurement file*" do LabVIEW, sendo uma utilizada para a gravação de dados provenientes das entradas analógicas do PIC e outra para a gravação de dados provenientes das entradas digitais do nosso PIC18F4550. Para efetuar a gravação dos dados coletados, basta um clique no botão "*REC Analog Data*" ou no botão "REC Digital Data", conforme a necessidade do usuário.

Na Figura 45 mostramos a seção do *Front Panel* responsável pela aquisição de dados digitais e analógicos provenientes de nossa plataforma didática.

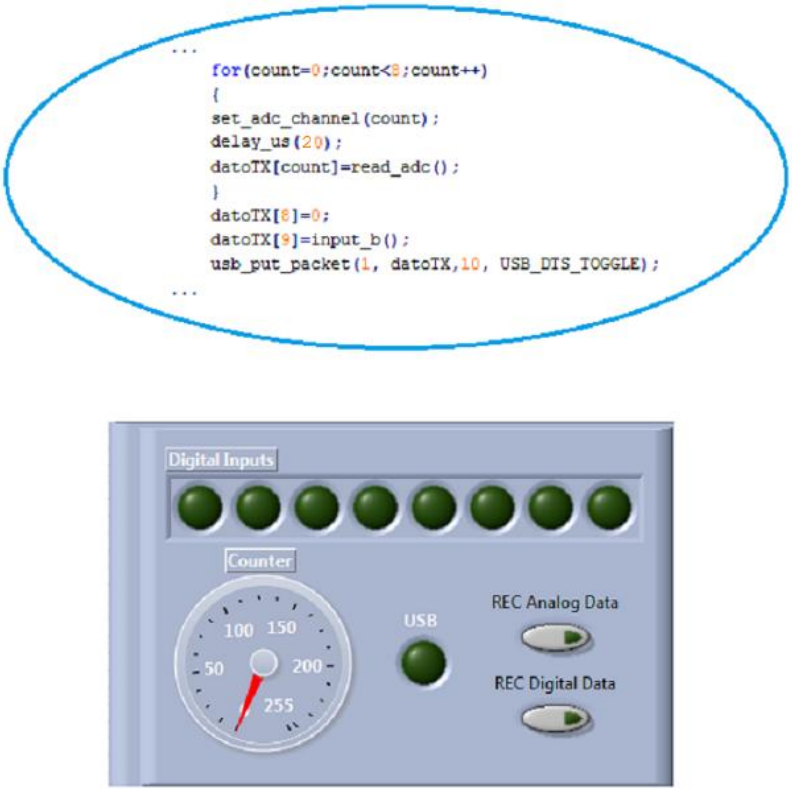


Figura 45 - Seção do *Front Panel* responsável pela transferência de dados da plataforma didática para o *host* e suas correlações com o *firmware* (Fonte: Os autores).

Para encerrar esta seção apresentaremos na Figura 46 o fluxograma de nossa VI principal para reforçar o entendimento dos detalhes que foram expostos.

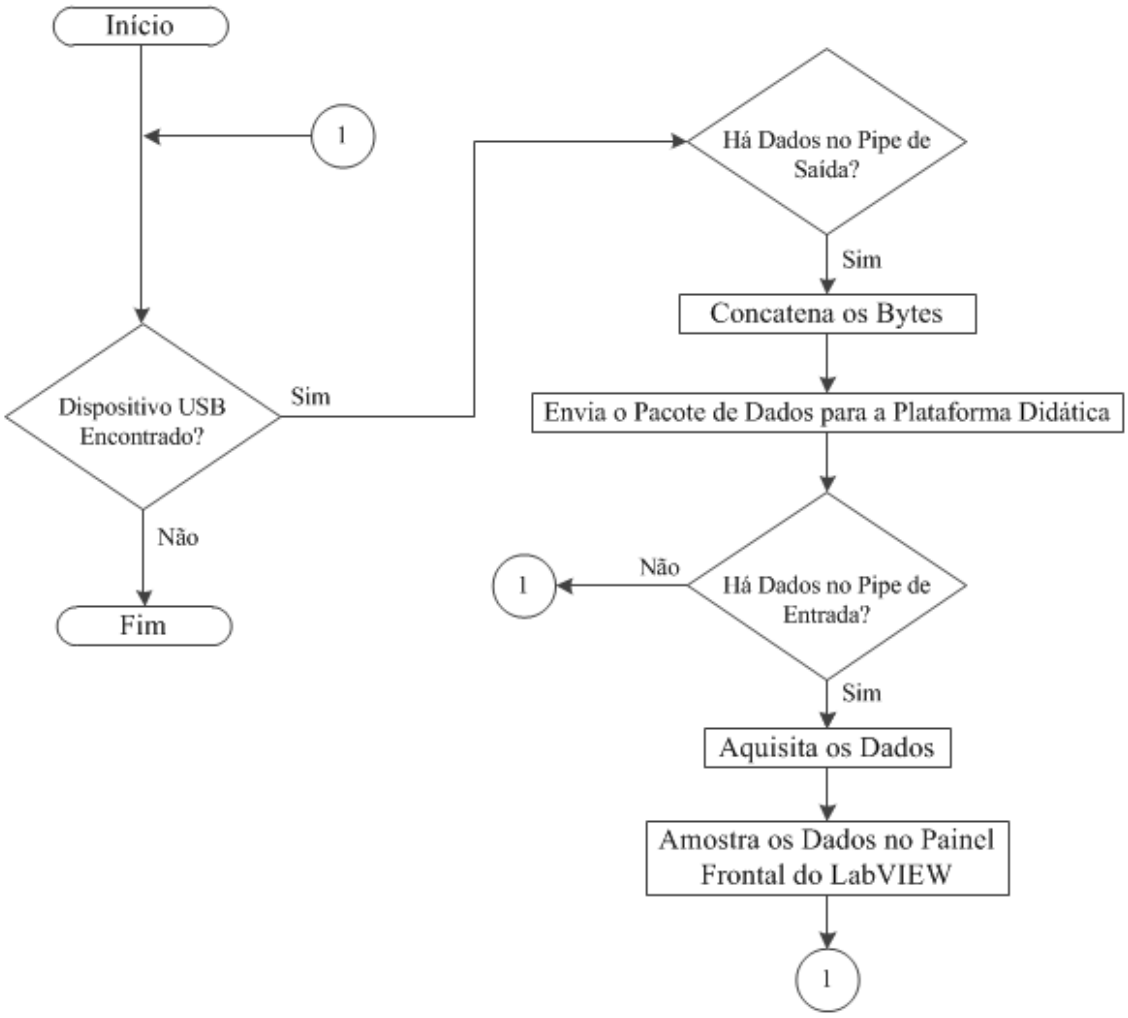


Figura 46 - Fluxograma da VI principal do projeto (Fonte: Os autores).

5.4 Processo de Enumeração da Plataforma Didática

Após as etapas descritas neste capítulo até a seção 5.3.2, partimos para o primeiro teste de detecção da nossa plataforma didática pelo *host*, que em nosso caso foi um *notebook* Dell Vostro i7 com sistema operacional Windows 8.1. Após realizarmos o *download* do *driver* mchpusb decidimos customizar o mesmo, modificando o arquivo mchpusb.inf, conforme Figura 47.

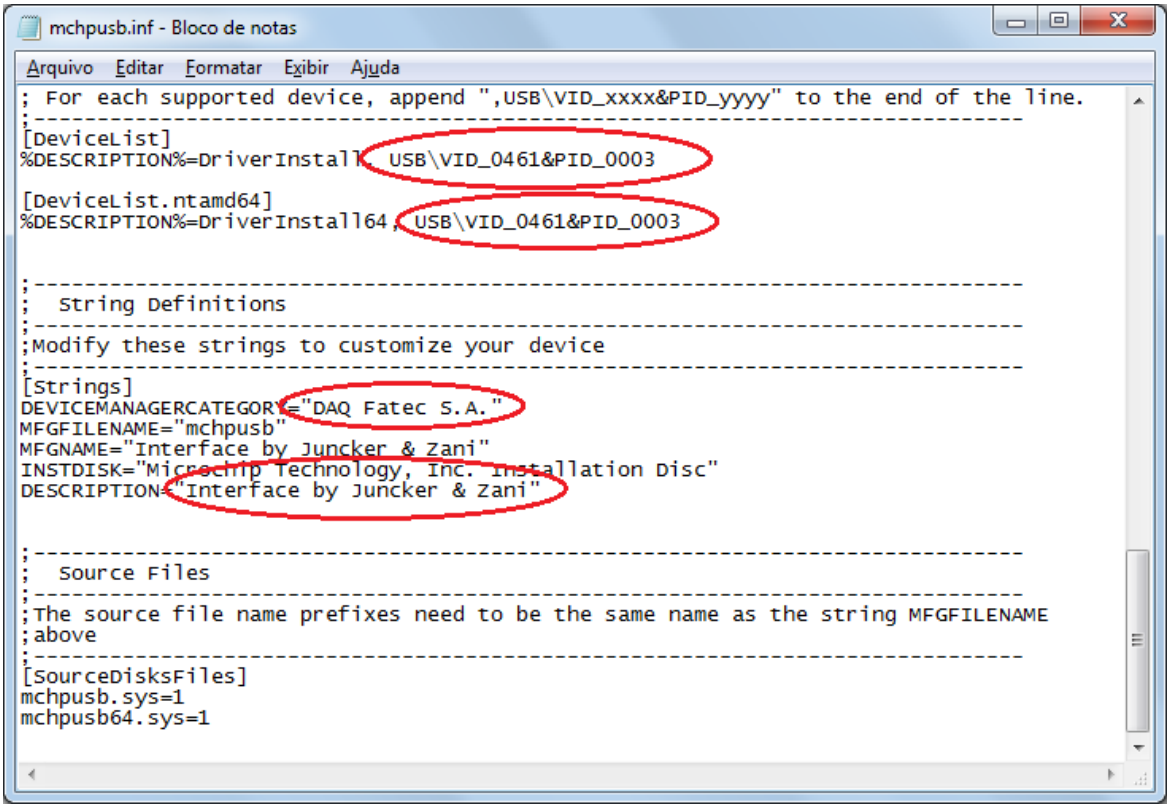


Figura 47 - Customização do *driver mchpusb* (Fonte: Os autores).

Outro ponto importante que pode ser verificado na imagem acima é o VID e PID que vem no "pacote" do *driver*. Os descritores contidos no *firmware* não podem diferenciar-se dos contidos no *driver*, caso contrário o *driver* irá se tornar incompatível e a aplicação não irá funcionar.

Neste teste inicial não ocorreu a enumeração de nosso dispositivo USB, mas sim um erro na fase de *loading* do *driver* mchpusb. Assim sendo, tivemos que auxiliar a enumeração de nosso dispositivo USB em sua primeira conexão, indicando em qual local o *host* deveria procurar o *driver* que customizamos. Na figura 48 é mostrado este erro do processo de enumeração da nossa plataforma didática USB , que pode ser visualizado no gerenciador de dispositivos do Windows.

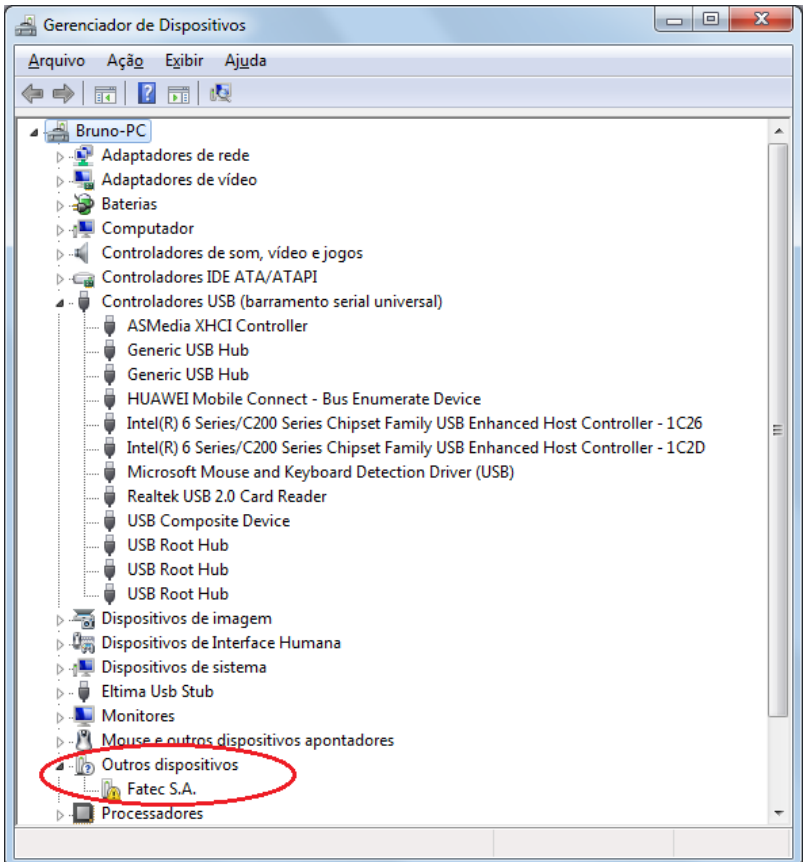


Figura 48 - Erro no processo de enumeração na primeira conexão do dispositivo USB (Fonte: Os autores).

A Figura 49 mostra como indicamos ao *host* o diretório onde ele poderia encontrar o *driver* mchpusb. Para isso clicamos com o botão direito do mouse em cima de nosso *driver* não reconhecido, depois clicamos em "Atualizar Driver" e posteriormente em "Procurar software de driver no computador".

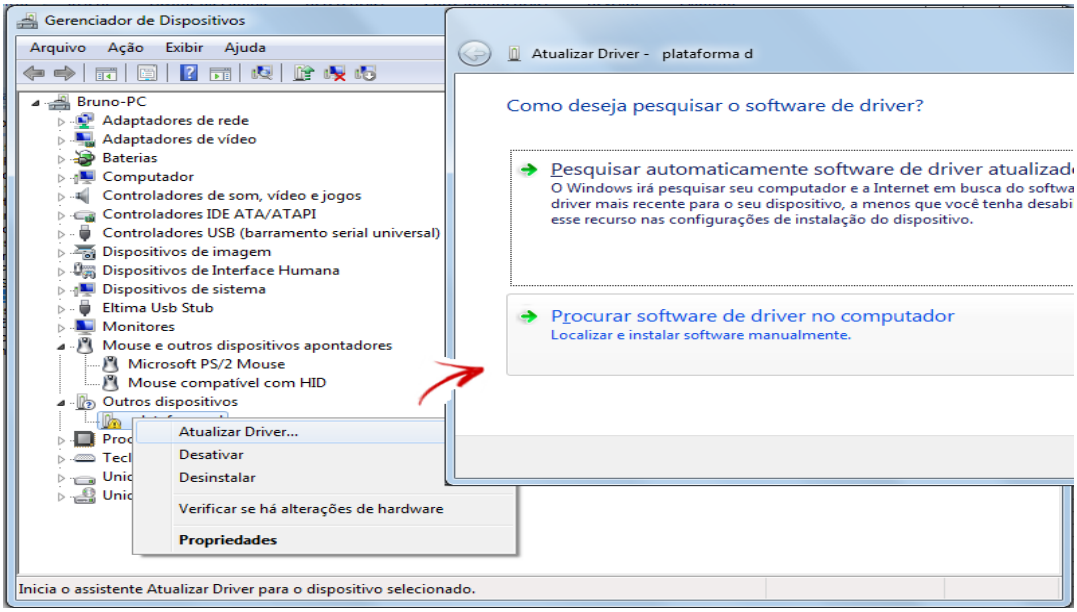


Figura 49 - Etapa de indicação do diretório de localização do *driver* mchpusb (Fonte: Os autores).

Após a instalação do *driver* o *host* detectou nossa plataforma e o processo de enumeração USB foi concluído como pode ser observado na Figura 50.

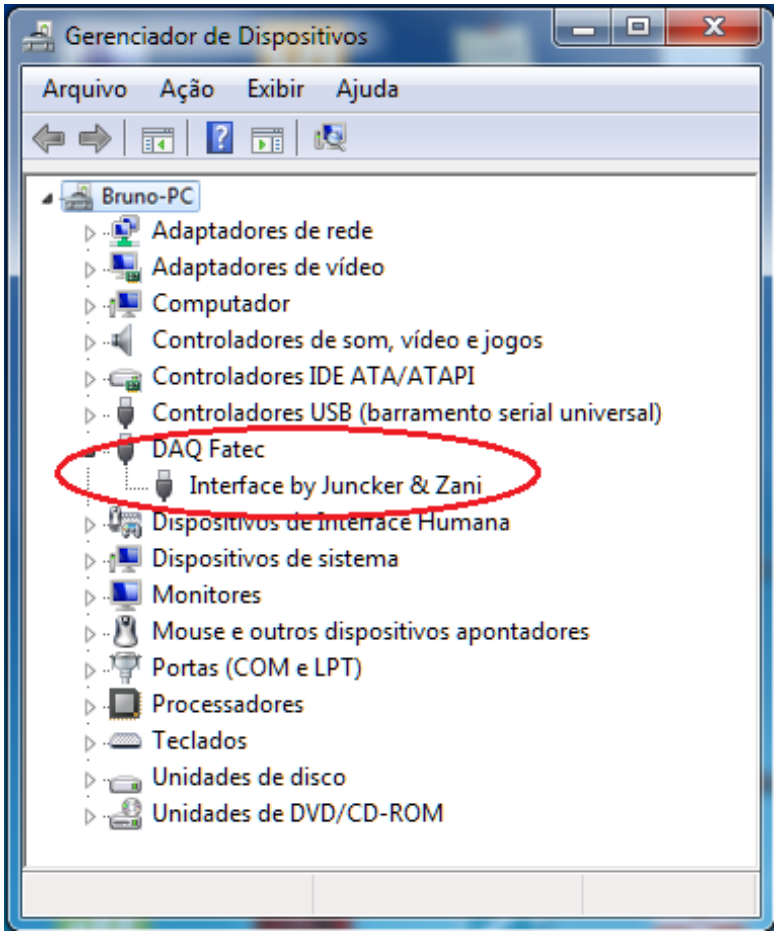


Figura 50 - Plataforma didática USB Detectada pelo *host* (Fonte: Os autores).

Depois desta primeira detecção o nosso dispositivo USB começou a funcionar no modo PnP normalmente, de forma a sempre reconhecer o *driver* em todas tentativas de conexões.

5.5 Resultados Obtidos

Após realizarmos a customização e todo o processo de enumeração do *drive*r de nosso dispositivo USB como citado na seção 5.4, partimos para os testes de aquisição e envio de dados com nossa plataforma didática. Nesta fase final decidimos adaptar nosso projeto com a inclusão de um display LCD na parte do *hardware*, já que muitos projetos da FATEC e outros em geral incluem este periférico como parte da interface física com o usuário. Desse modo a VI principal sofreu algumas alterações que não foram detalhadas na seção 5.3.2, mas que serão abordadas nas seções 5.5.1 à 5.5.3.

Primeiramente apresentaremos a parte de envio de dados do *host* para a plataforma didática, sendo que o usuário que comanda este envio de dados atraváes da interface LabVIEW. Em seguida apresentaremos os resultados das aquisições dos sinais enviados da plataforma para o *host*. Para encerrar mostraremos a montagem final do *hardware* do projeto.

5.5.1 Envio de Dados do *Host* para a Plataforma Didática

Inicialmente, apresentaremos o envio do sinal modulado em largura de pulso (PWM), sendo que a modulação do pulso é comandada pelo usuário através do *Front Panel* do LabVIEW.

Na Figura 51 é possível perceber a variação da luminosidade dos LEDs da plataforma didática de acordo com a variação do pulso PWM comandado pelo usuário através deos *knobs* do *Front Panel* de nossa VI.

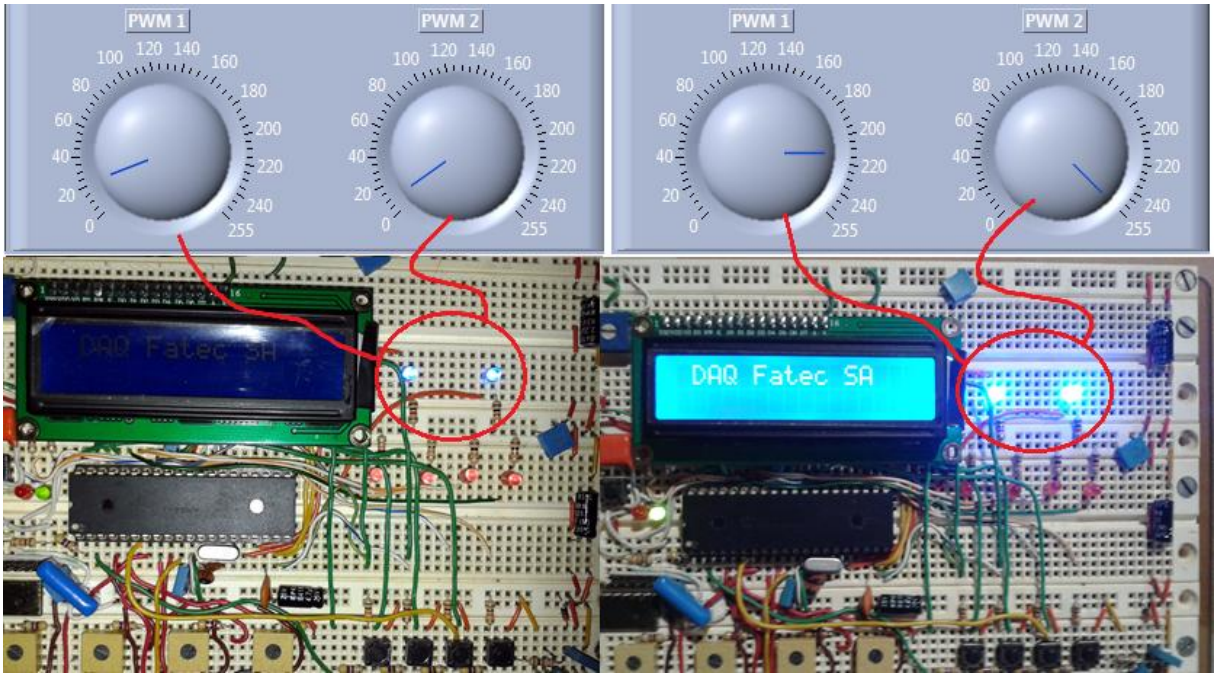


Figura 51 - Resultado do sinal PWM enviado pelo *host* (Fonte: Os autores).

O *Host* também é responsável por enviar os comandos digitais para acionar os LEDs conectados ao PortB da plataforma didática, como pode ser verificado na Figura 52.

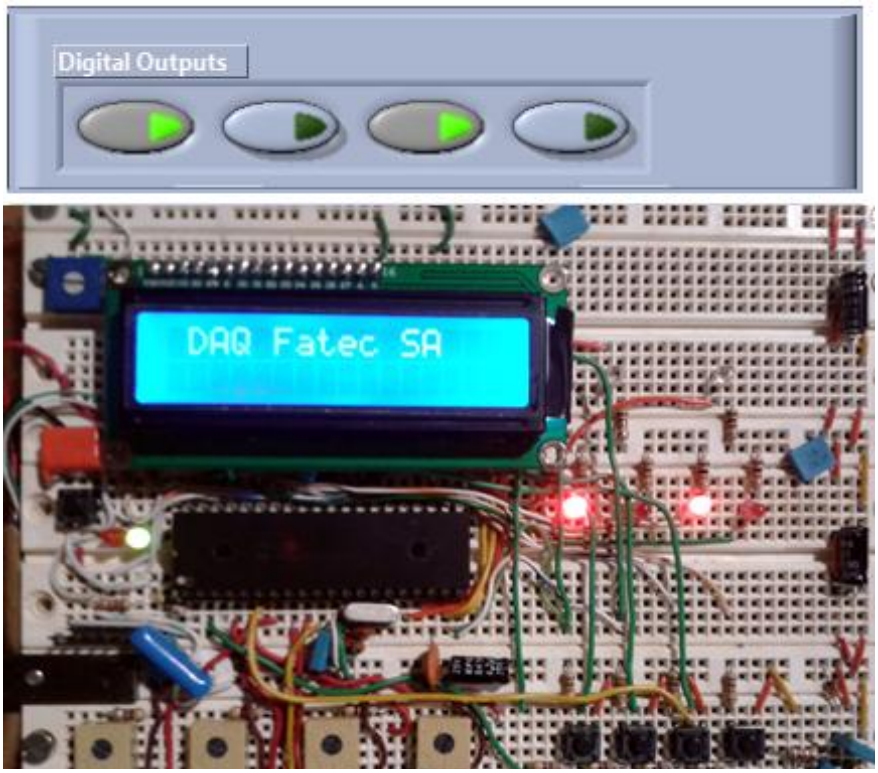


Figura 52 - Comandos digitais enviados via LabVIEW para a plataforma didática (Fonte: Os autores).

5.5.2 Envio de Dados da Plataforma Didática para o *Host*

A plataforma didática desenvolvida neste trabalho possibilita a leitura de sinais analógicos de baixa freqência. Na Figura 53 mostramos o resultado da conversão AD no *Front Panel* de nossa VI do sinal de dois potenciômetros ligados ao PortA do PIC da plataforma.

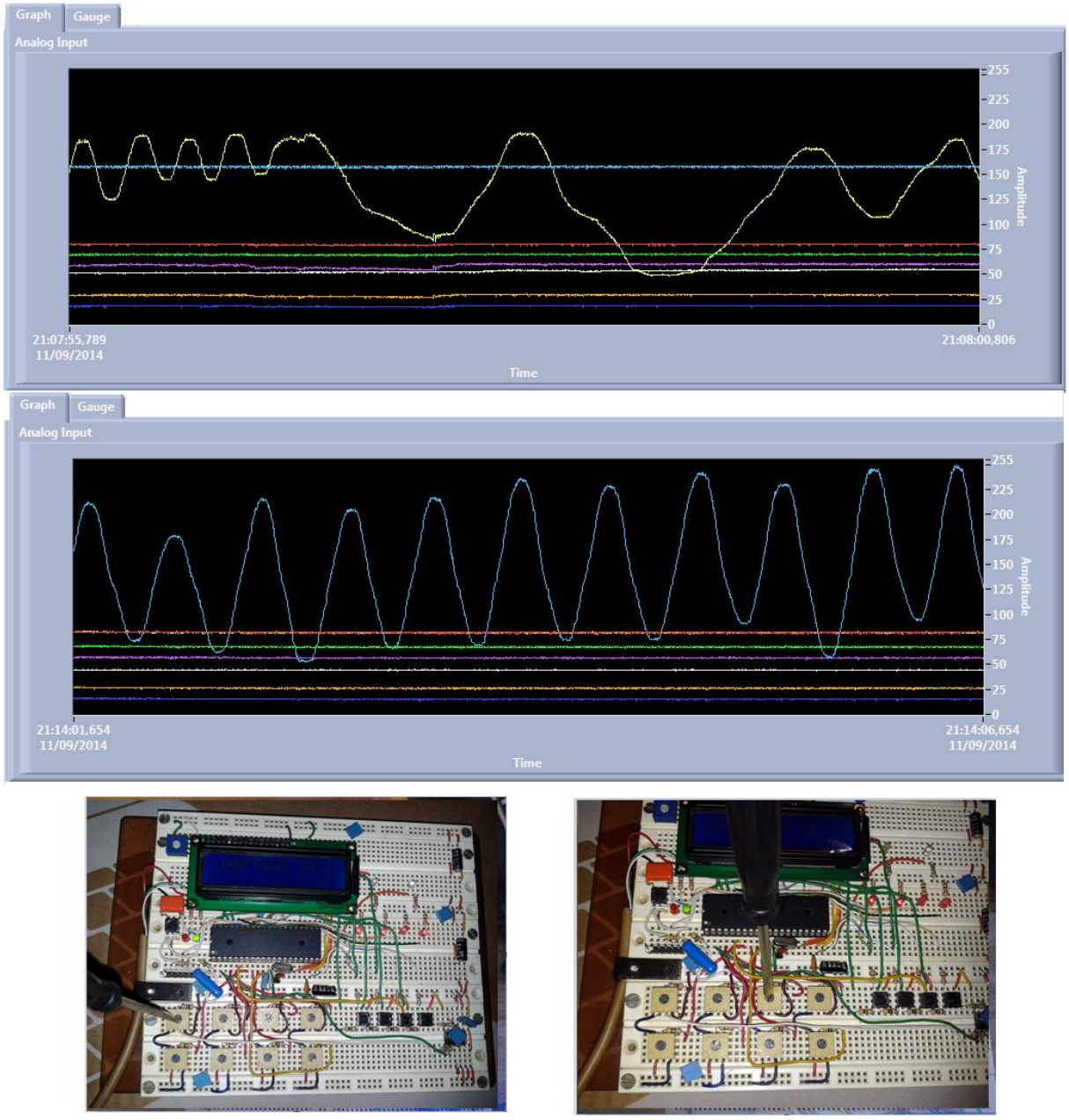


Figura 53 - Resultado de conversões AD no PORTA do PIC que estão sendo enviados via USB para o *host* (Fonte: Os autores).

Também temos como parte dos resultados os comandos digitais enviados dos *push-buttoms* que também estão conectados o portB do PIC18F4550. Ao pressionar algun deles fisicamente, imediatamente o LED no *Front Panel* correspondente irá acender, conforme ilustrado na Figura 54.

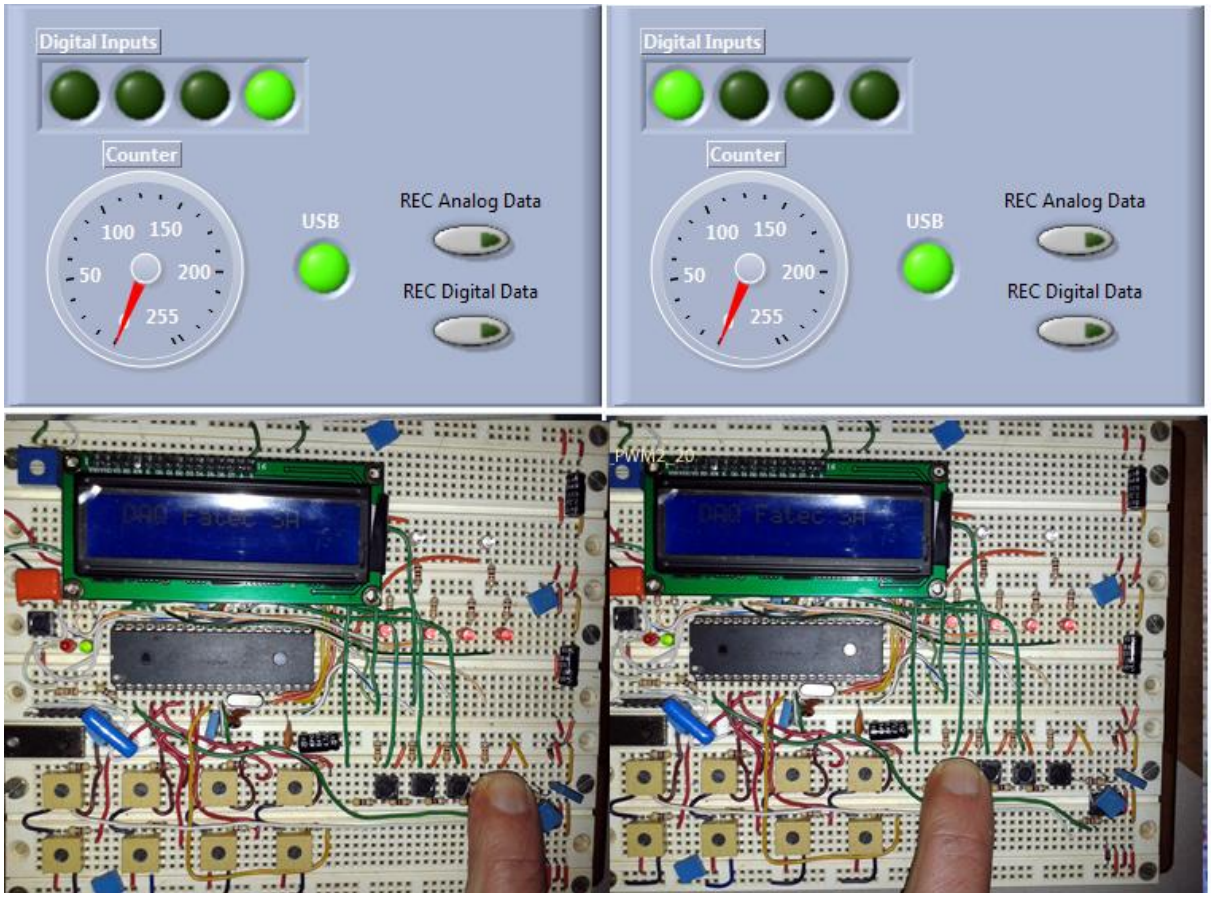


Figura 54 - Comandos digitais enviados da plataforma didática para o *host* (Fonte: Os autores).

Um outro dado que pode ser observado via LabVIEW é o retorno do valor da contagem do Timer0 do PIC, sendo este ativado com um simples fio conectado ao pino 6 do PIC18F4550 (*Timer 0 external clock input*). Na Figura 55 é ilustrado um valor aleatório desta contagem.

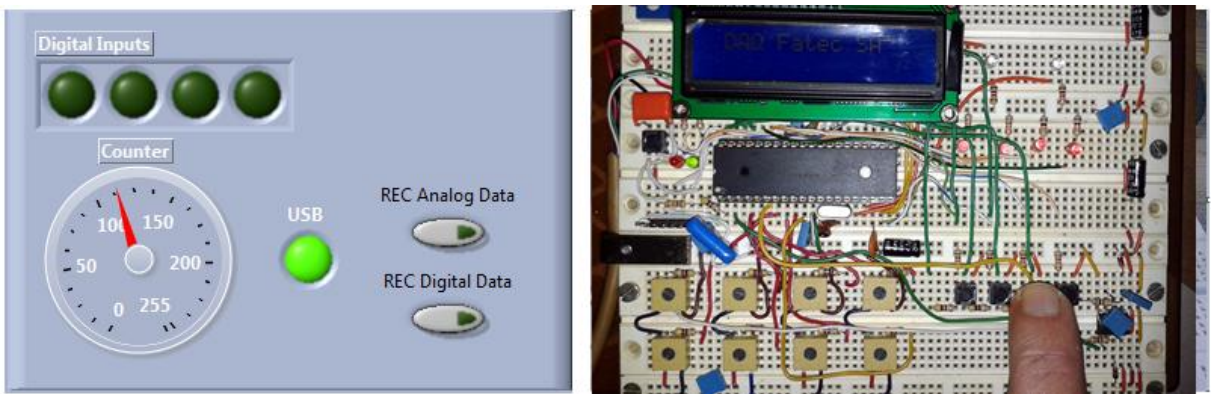


Figura 55 - Envio do valor da contagem do Timer0 do PIC para o *host* (Fonte: Os autores).

5.5.3 Display LCD e Aquisição dos Dados

Nas imagens 56 e 57 apresentaremos a resultado obtido com a implementação do dysplay LCD no modo *4 Bit Mode* e um exemplo de aquisição de dados ao pressionar algum dos botões "REC" no *Front Panel* do LabVIEW. Apresentamos como exemplo do dysplay a simples frase "DAQ Fatec SA".

Figura 56 - Resultado da implementação do *display* LCD no modo *4Bit Mode* (Fonte: Os autores).

Em relação ao exemplo de aquisição apresentaremos um fragmento de um dos arquivos gerados ao apertar o botão "REC Digital Data" no *Front Panel* do LabVIEW.

Pode-se perceber que no arquivo gerado são gravados milhares de iterações. No quadrado em destaque a esquerda visualizamos as posições do array e no quadrado em destaque a direita podemos ver o valor booleano contido nas posições do array.

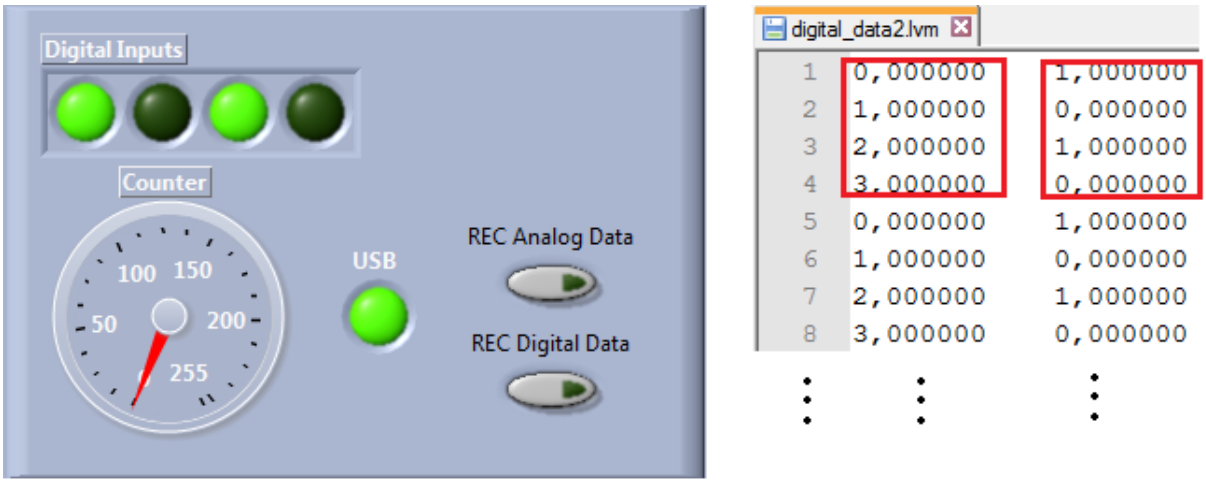


Figura 57 - Exemplo de arquivo gerado na aquisição de dados digitais (Fonte: Os autores).

5.5.4 Limitações do projeto

A principal limitação que encontramos ao longo do desenvolvimento desta plataforma didática foi em relação à taxa de aquisição de dados de sinais com frequências elevadas. Mesmo após muitos testes com a nossa plataforma USB não conseguimos identificar onde estava ocorrendo o erro. Inicialmente pensamos que o equívoco era em relação à conversão analógico-digital, então mudamos o nosso *firmware* principal e adcionamos um tratamento por interrupção do Timer2 do PIC 18F4550, configurando-o com uma interrupção a cada 250 microsegundos. A conversão ocorreria a cada 40 interrupções do Timer2 (10 milisegundos), o que em teoria, pelo teorema de Nyquist, daria para aquisitar sinais com frequências de até 50Hz sem perda de dados e sem a ocorrência do fenômeno *aliasing*, já que nossa taxa de aquisição de dados seria de 100Hz.

```
...
#int_timer2
void timer2(void) //250us
{
   if (time_ADC) time_ADC--;
   if (!time_ADC)
   {
      for(count=0;count<8;count++)
      {
          set_adc_channel(count);
          delay_us(20);
          time_ADC = 40;
          dataTX[count]=read_adc();
      }
   }
}
...
```

```
void main(void)
{
   ...
   setup_timer_2(T2_DIV_BY_1, 200, 15);
   ...
}
```

Figura 58 - Fragmentos do *firmware* principal da configuração do Timer2 utilizados na tentativa de melhorar a aquisição de sinais com frequência elevada (Fonte: Os autores).

Na figura 58 pode-se visualizar um fragmento do nosso *firmware* principal responsável pela interrupção do timer2. Mas mesmo com essas configurações não obtivemos muito sucesso: o máximo que conseguimos foi aquisitar sinais analógicos com frequência de até 10Hz, como ilustra a Figura 59.

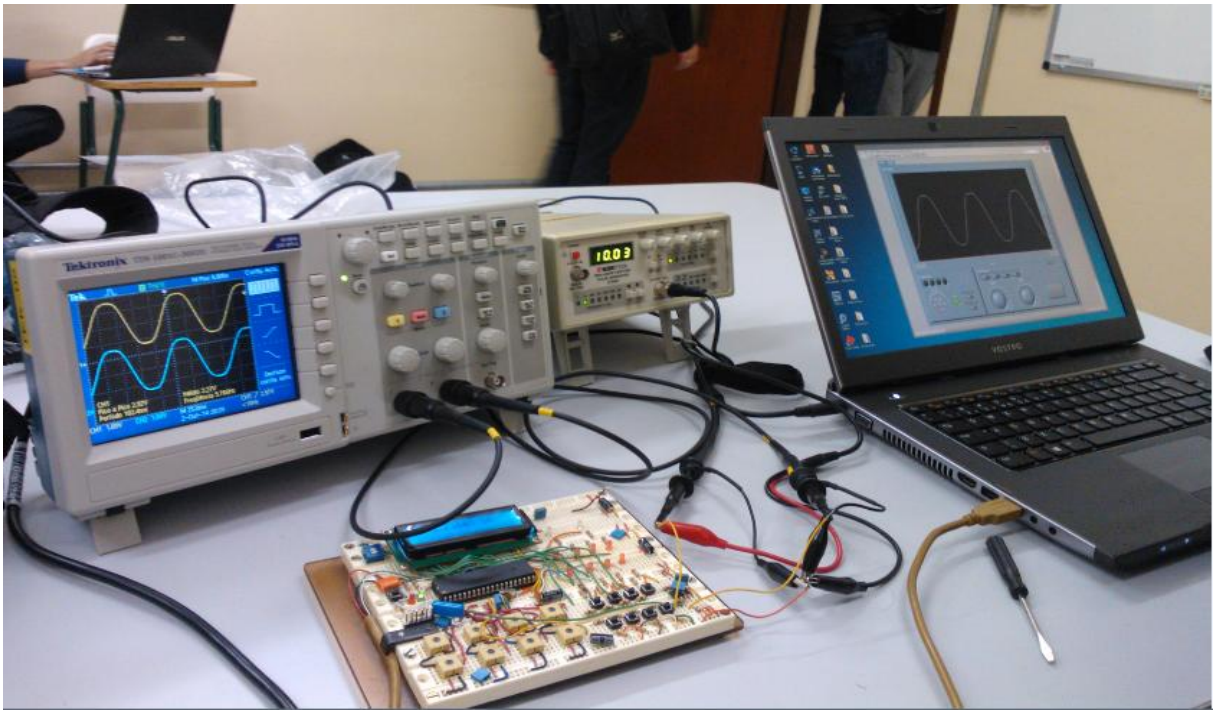


Figura 59 – Visão geral do teste de aquisição de uma onda senoidal com frequência de 10 Hz (Fonte: Os autores).

Já na Figura 60 e 61 é possível perceber que um sinal com uma frequência acima de 10 Hz começa a limitar a integridade dos dados que serão coletados pela plataforma didática, apresentando pequenas deformações no sinal apresentado no "*waveform graph*" do LabVIEW.

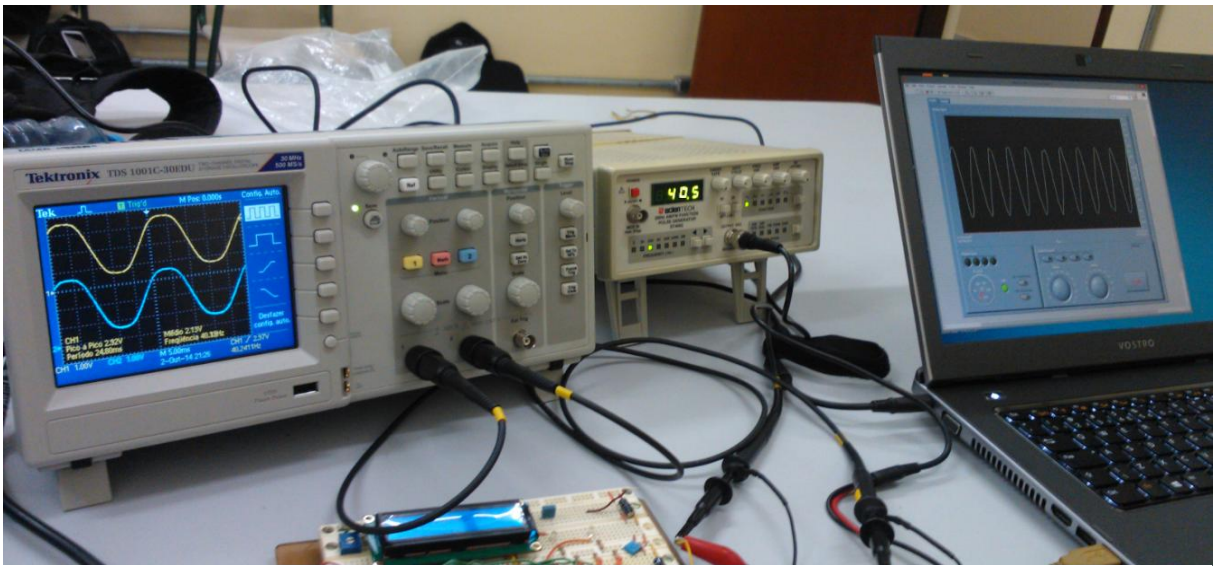


Figura 60 - Visão geral do teste de aquisição de uma onda senoidal com frequência de 40hz (Fonte: Os autores).

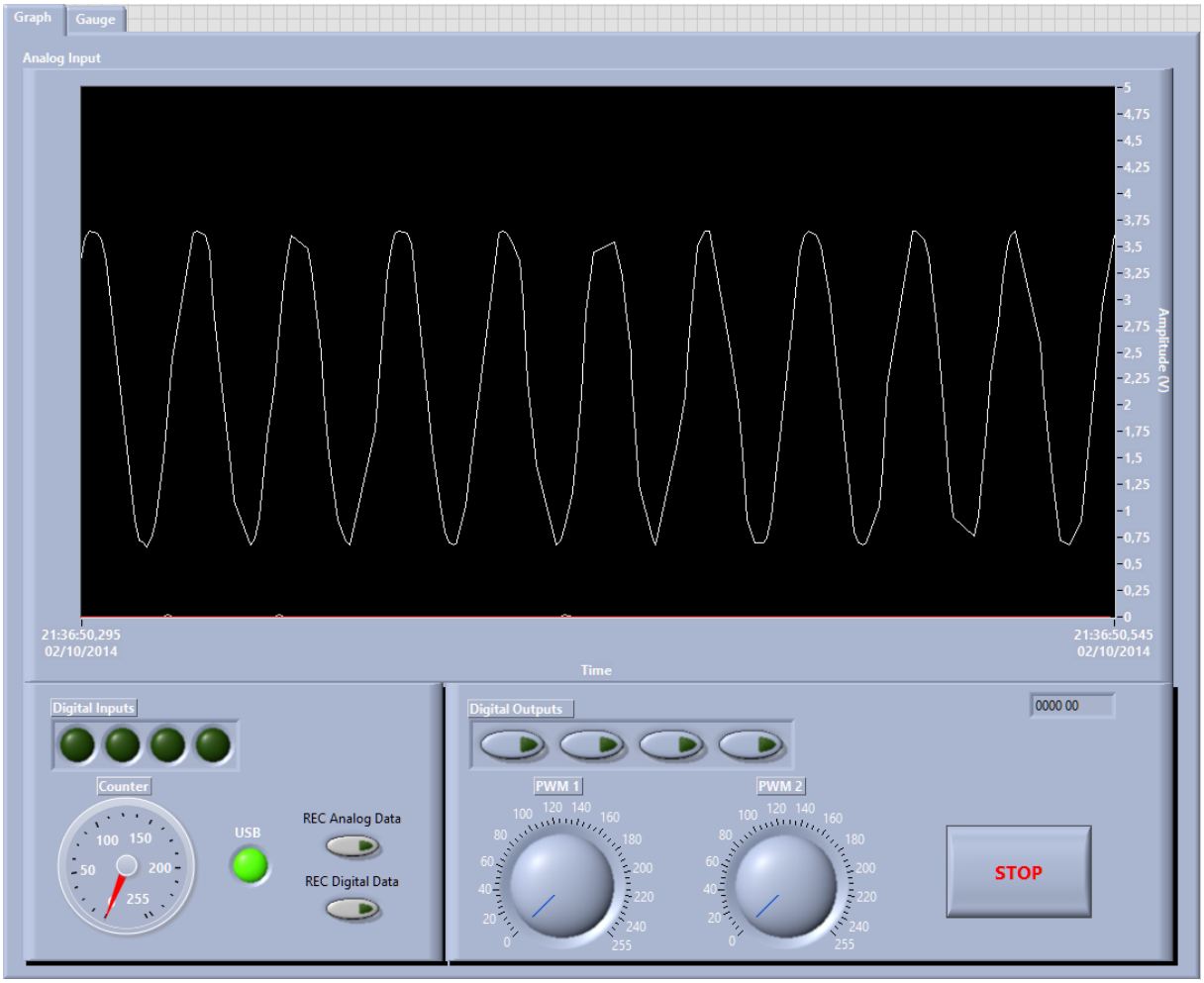


Figura 61 - Visão ampliada do teste de aquisição de uma onda senoidal com frequência de 40 Hz (Fonte: Os autores).

Na visão ampliada do teste que foi realizado com uma onda senoidal com 40hz de frequência emitida por um gerador de sinais, fica claramente visível as deformações citadas anteriormente.

A partir de então começamos a fazer novos testes na parte de *firmware*, como por exemplo reduzir os canais analógicos que estavam sendo utilizados no projeto para efetuar as conversões analógico-digitais (apesar do PIC 18F4550 possuir diversos canais para conversão, ele possui apenas um ADC interno). Em teoria isso também elevaria nossa capacidade de aquisição de dados. Mas novamente o resultado foi o mesmo, como já citado anteriormente. Na figura 62 pode ser visualizado a parte do *datasheet* relacionada com o ADC interno do PIC 18F4550 que nos levou a realizar este teste de redução dos canais analógicos.

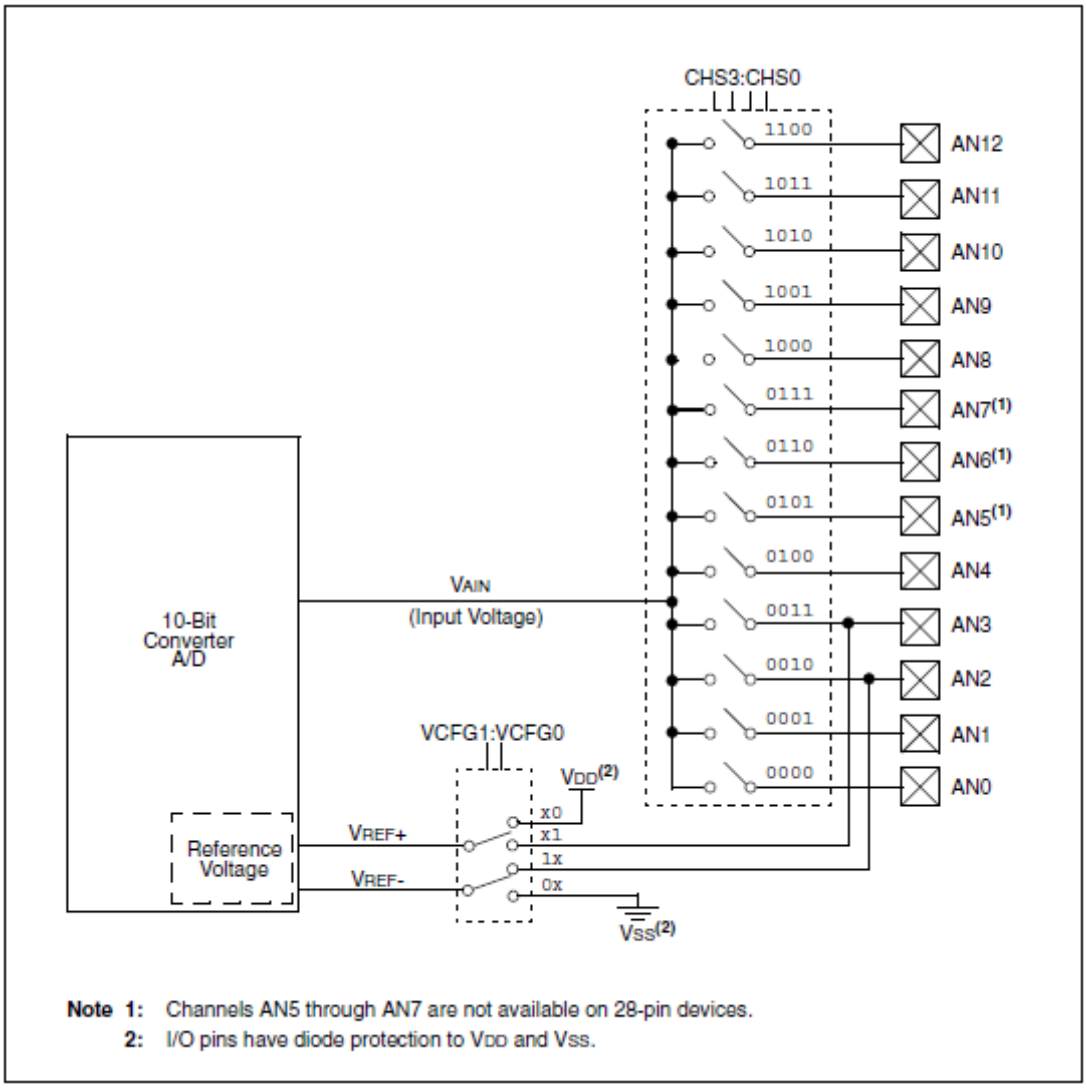


Figura 62 - Diagrama do ADC interno do PIC 18F4550 [Extraído de (MICROCHIP ® *datasheet*, 2006)].

Depois disso realizamos outras tentativas para corrigir o erro na parte do *firmware*, como preencher os 64 *bytes* de cada pacote de dados que é enviado pelo PIC via USB, tentativas de integrar um *buffer* circular à interrupção de modo a enviar as 64 posições da variável vetor do nosso *firmware* de uma forma que estas posições sejam "sobrepostas" no momento correto, ou seja, após o envio do pacote de dados via USB. A figura 63 mostra fragmentos de nosso *firmware* principal com uma dessas tentativas de integrar o *buffer* circular à interrupção gerada pelo PIC (neste teste foi utilizado a interrupção pelo Timer1).

```
...
    if (cem_microsegundos > 10)
    {
        count++;
        set_adc_channel(1);
        delay_us(20);
        dataTX[count]=read_adc();
        cem_microsegundos = 0;
    }
        if    (count > 64)
        {
        count=0;
        usb_put_packet(1, dataTX,64, USB_DTS_TOGGLE);
        }
...
```

```
...
#int_timer1
void timer1(void)
  {
   set_timer1(64336 + get_timer1()); //100us
   cem_microsegundos++;
  }
 ...

void main(void)
{
...
   setup_timer_1(T1_INTERNAL | T1_DIV_BY_1);
...
}
```

Figura 63 - Fragmentos do *firmware* principal da configuração do Timer1 utilizados na tentativa de melhorar a aquisição de sinais com frequência elevada com a integração de um *buffer* circular (Fonte: Os autores).

Após estas e outras modificações no *firmware* resolvemos estudar a nossa interface gráfica via LabVIEW com a finalidade de resolver o problema. Tentamos também utilizar outro *driver*, o WinUSB, desenvolvido pela Windows, mas no fim acabou não fazendo diferença em relação ao aperfeiçoamento da aquisição de sinais analógicos.

6 Discussões Finais e Conclusão

"Não devemos parar de explorar. E o fim de toda nossa exploração será chegar ao ponto de partida e ver o lugar pela primeira vez."

Stephen R. Covey

Discutiremos neste capítulo algumas propostas para futuras implementações para aperfeiçoamento e continuidade do projeto, assim com as nossas considerações finais sobre a plataforma didática desenvolvida.

6.1 Propostas Futuras de Continuidade e Possíveis Melhorias do Projeto

Para aqueles que estiverem realizando estudos relacionados com o projeto desenvolvido e ocorrer o interesse na continuidade deste trabalho, segue abaixo algumas sugestões para melhorias e algumas propostas futuras:

- Realizar um estudo mais aprofundado sobre sistemas de aquisição de dados via USB, visando melhorar as limitações apresentadas na seção 5.5.4 e tendo como proposta um desenvolvimento de um *data logger*, com objetivos diferentes de uma plataforma didática como a que foi apresentada neste trabalho;
- Utilizar uma memória externa para armazenar os dados coletados;
- Utilizar uma metodologia diferente para medir a qualidade dos dados coletados e não somente de forma visual, analisando harmonicamente as distorções que ocorrem à medida que a frequência do sinal que é coletado aumenta.. Também pode ser feito uma avaliação do sinal por meio de erro médio quadrático nesta nova metodologia;
- Medir o tempo de ciclo total de um scan do programa que é executado pelo PIC, pois se este tempo for crítico pode interferir no processo de conversão analógica;
- Outra proposta de continuidade do projeto também seria o desenvolvimento de um osciloscópio USB com interface gráfica para utilização do usuário via LabVIEW, podendo ser utilizado um microcontrolador mais robusto para tal;
- Talvez seja possível a criação de uma interface gráfica de relacionamento com o usuário via *smartphone*, ou seja, o desenvolvimento de um aplicativo de celular capaz aquisitar ou enviar dados via USB para a plataforma didática.

6.2 Conclusão

O desenvolvimento deste projeto trará uma oportunidade de aprendizado para os alunos da FATEC Santo André e estudantes em geral, trazendo por final uma plataforma didática que apresenta uma estratégia de comunicação simplificada, com *hardware*, *firmware*, *driver* e linguagem G de fácil manipulação. O modo *Bootloader*, além de tornar a programação do PIC direta em memória via USB, também acaba por acelerar o processo de aprendizado, pelo fato de poupar tempo na programação como também poupar o uso de recursos da faculdade, tornando-se uma ferramenta extremamente útil para o aluno. Outro aspecto que vale a pena salientar nesta conclusão é o baixo custo do projeto como um todo (em torno de R$126,00 em março de 2014), como pode ser observado através do Anexo 8.2. A proposta deste trabalho apresenta uma alternativa que proporciona flexibilidade de programação (tanto do lado do LabVIEW como do lado do PIC), permitindo fácilmente alterações dos sinais que são enviados da plataforma para o *host* e vice-versa, como também facilita alterações da interface gráfica pré-definida.

7 Referências Bibliográficas

ALECRIM, Emerson (2009), **Tecnologia USB (Universal Serial Bus).** Disponível em: <http://www.infowester.com/usb.php>. Acessado em 10 de Janeiro de 2014.

AMORIM, Marcelo Bezerra (2008), **Uma Biblioteca para Comunicação com a Camada Física USB Padrão ULPI,** Monografia, Universidade Federal de Pernambuco, Pernambuco.

AXELSON, Jan. **USB Complete: Everything You Need to Develop Custom USB Peripherals.** 3ª ed. Wisconsin: Lakeview Research, 2001.

BAPTISTA, Manuel (2006), **Sistemas de Aquisição de Dados,** Relatório, Escola Superior de Tecnologia Viseu, Portugal.

BARBOSA, Tiago Silva (2008), **Desenvolvimento de um Sistema de Monitoramento de Máquinas de Indução Trifásicas Visando Assimetrias de Rotor em Gaiola,** Monografia, Universidade Federal do Espírito Santo, Espírito Santo.

CEARÁ, Jonis Maurin (2014), **PIC + USB + Bootloader = fácil.** Disponível em: <http://www.jonis.com.br/index.php/13-eletronica/14-pic-usb-bootloader-facil>. Acessado em 23 de Junho de 2014.

CEBRAC (2012), **O que é USB e porque usar dispositivos com esta tecnologia?.** Disponível em: <http://cebracms.com.br/aula/o-que-e-usb-e-porque-usar-dispositivos-com-esta-tecnologia/#.U6YwJLGGfS5>. Acessado em 21 de Junho de 2014.

CORPORATION**,** Compaq Computer. et al. **Universal Serial Bus Specification.** Revisão 1.1, 1998.

CUETO, J. A. Pérez; ESTRADA, F. R. López (2009), **Comunicación USB de Alta Velocidad Entre LABVIEW y um Microcontrolador para La Adquisición de Datos em Tiempo Real.,** Artigo, Instituto Tecnológico de Tuxtla Gutiérrez, México.

DOGAN, Ibrahim. **Advanced PIC Microcontroller Projects in C.** 1ª ed. Massachusetts: Elsevier, 2008.

ERDOGĂN, Balkar, **Tutorial 1: Implementation of a USB based PIC-to-PC communication,** Artigo, METU-EEE, Turquia, 2011.

FILHO, Clidenor (2005), **Redes Industriais – parte 1,** Artigo, UNIUBE, Brasil.

HENRIQUE, Tiago (2013), **PIC: Bootloader USB HID.** Disponível em: <http://microcontrolandos.blogspot.com.br/2013/09/pic-bootloader-usb.html>. Acessado em 24 de Junho de 2014.

KITANI, Edson Caoru (2013), **Notas de aula da disciplina Ferramentas Computacionais** – FATEC Santo André, São Paulo.

LabVIEW ® Manual (2006), **Using the LabVIEW Run-Time Engine.** Disponível em: <http://zone.ni.com/reference/en-XX/help/371361B-01/lvhowto/using_the_lv_run_time_eng/>. Acessado em 13 de Fevereiro de 2014.

MESSIAS, Antonio Rogério (2013), **Curso USB/Serial – Controle de Dispositivos.** Disponível em: <http://www.rogercom.com/CursoOnlineUSB/ModuloUnicoAula005.htm>. Acessado em 25 de Janeiro de 2014.

MICROCHIP ® (2006), **Datasheet do microcontrolador PIC 18F4550.** Disponível em: <http://ww1.microchip.com/downloads/en/devicedoc/39632c.pdf>. Acessado em 26 de Junho de 2014.

MICROCHIP ® (2014), **Microchip Libraries for Applications.** Disponível em: <http://www.microchip.com/pagehandler/en-us/devtools/mla/home.html>. Acessado em 10 de Julho de 2014.

Microchip. **USB: Implementação de Hardware e Software.** 11º Seminário Técnico Microchip Masters Brasil, 2006.

NATIONAL INSTRUMENTS, **Como as Pessoas Estão Usando os Dispositivos de Aquisição de Dados da National Instruments.** Disponível em: <http://www.ni.com/data-acquisition/applications/pt/>. Acessado em 01 de Março de 2014.

PEDREIRAS, Paulo (2010), **Notas de aula da matéria de Sistemas de Tempo Real** – Universidade de Aveiro, Portugal.

RAMIRES, Leandro Santiago; MURASUGI, Márcio Tetsuya (2003), **Biblioteca de Aquisição de Dados,** Monografia, Universidade Braz Cubas, São Paulo. [20]

RBFAUTOMAÇÃO (2013), **O que é LabVIEW?** Disponível em: <http://www.rbfautomacao.com.br/o-que-e-labview.html>. Acessado em 15 de Março de 2014.

SALAS, Jose (2013), **Principales características del Pic 18F4550,** Disponível em: <http://todoelectrodo.blogspot.com.br/2013/02/pic-18f4550.html>. Acessado em 27 de Junho de 2014.

SANTOS, Leonardo de Sá Leal (2009), **Sistema de Comunicação USB com Microcontrolador,** Monografia, Universidade de Pernambuco, Pernambuco.

SOUZA, Vitor Amadeu (2010), **Tutorial USB.** Disponível em: **<**http://www.cerne-tec.com.br/artigo%20USB.pdf>. Acessado em 02 de Fevereiro de 2014.

STADLER, Christian (2011), **Serial PIC Bootloader.** Disponível em: **<**http://www.picprojects.net/serialbootloader/>. Acessado em 24 de Junho de 2014.

STALLINGS, William. **Arquitetura e Organização de Computadores.** 5ª ed. São Paulo: Prentice Hall, 2002.

ULLRICH, Freddy Albinus (2003), **Protótipo de um Hardware Periférico para Mixagem de Músicas MP3 via Saída U.S.B.,** Monografia, Universidade Regional de Blumenau, Santa Catarina.

U**SB e Bootloader.** Disponível em: **<**http://www.hpspin.com.br/site1/bootloader/>. Acessado em 23 de Junho de 2014.

VTM GROUP **(**2014), **USB Class Codes.** Disponível em: <http://www.usb.org/developers/defined_class>. Acessado em 30 de Setembro de 2014.

Virtual Instrumentation With LabVIEW, http://www.docstoc.com/docs/30543443/LABVIEW-INTRODUCTION, acessado em 25 de Junho de 2014.

WIKPEDIA (2014), **LabVIEW.** Disponível em: <http://en.wikipedia.org/wiki/LabView>. Acessado em 27 de Fevereiro de 2014.

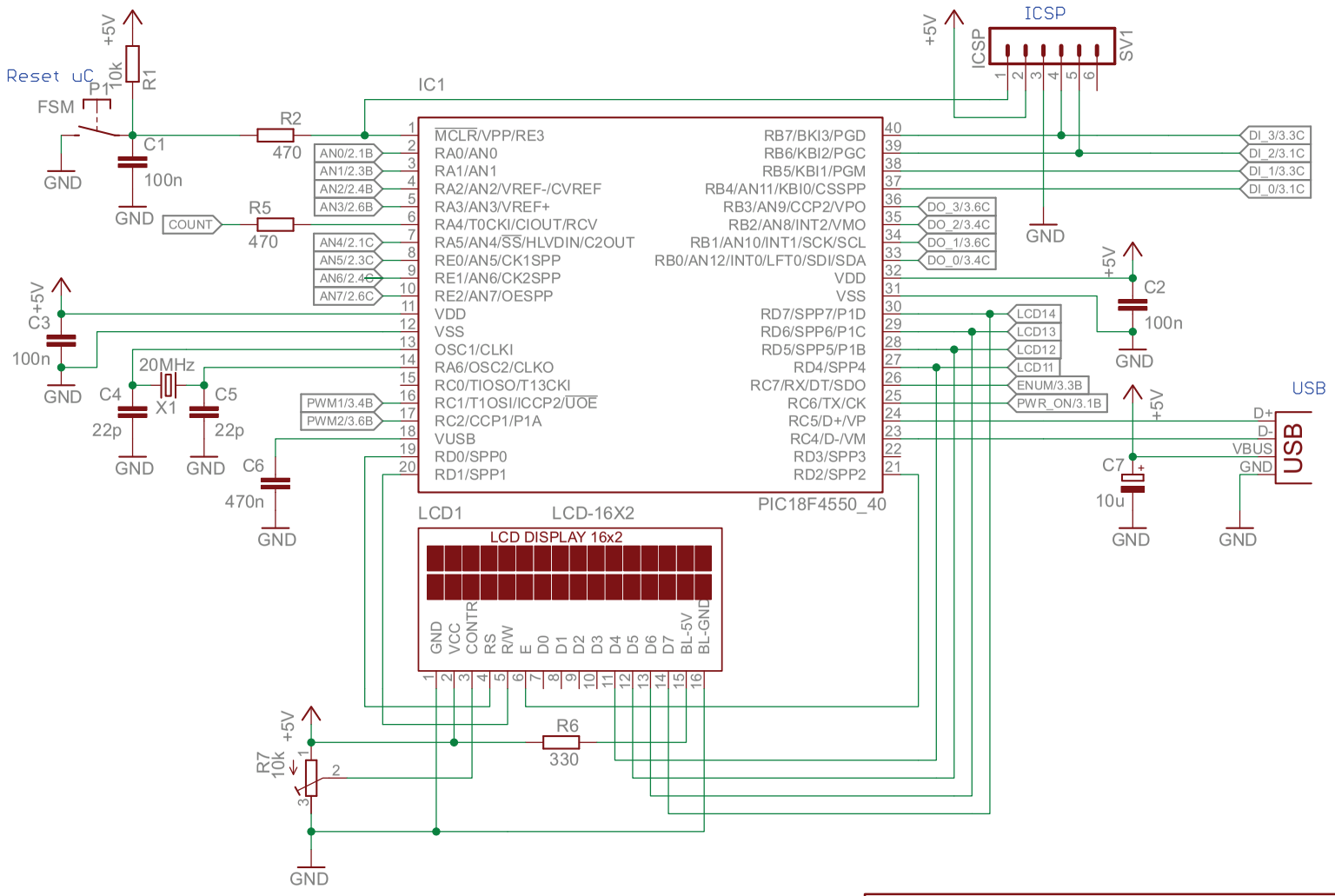
ZEMBOVICI, Kleiton Chochi; FRANCO, Marcelo Gonçalves (2009), **Dispositivo para Aquisição de Sinais e Controle Digital via USB,** Monografia, Universidade Federal do Paraná, Paraná.

8 Anexos

8.1 Esquema Eletrônico da Montagem do *Hardware*

Interface BR 2014

MICROCONTROLLER, USB, DISPLAY LCD, ICSP.



USB CONNECTION

PIN25 - PWR_ON
PIN26 - ENUM

ANALOG IMPUT

PIN2 - AN0
PIN3 - AN1
PIN4 - AN2
PIN5 - AN3
PIN7 - AN4
PIN8 - AN5
PIN9 - AN6
PIN10 - AN7

DIGITAL INPUT

PIN37 - DI_0 (Bootloader)
PIN38 - DI_1
PIN39 - DI_2
PIN40 - DI_3
PIN6 - COUNT

DIGTAL OUTPUT

PIN33 - DO_0
PIN34 - DO_1
PIN35 - DO_2
PIN36 - DO_3

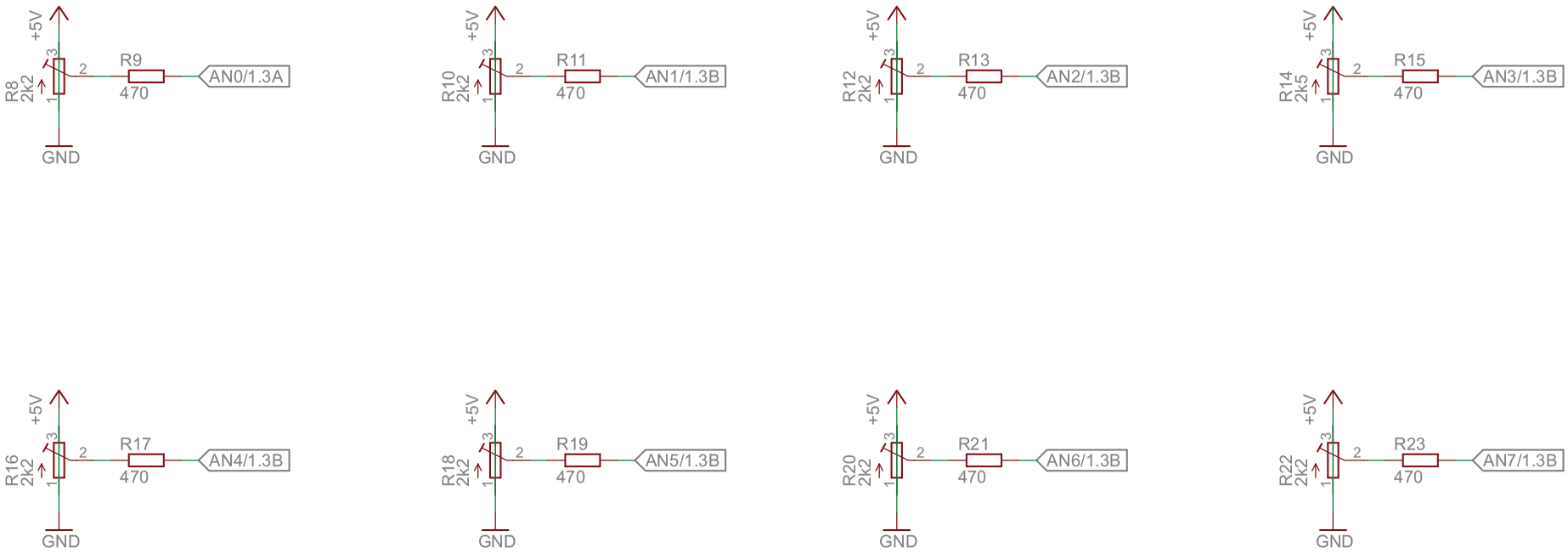
PWM OUTPUT

PIN19 - PWM1
PIN20 - PWM2

FATEC - Tecnologia em Eletrônica Automotiva	
Interface_BR/USB/LabVIEW	Bruno Zani Sampaio
Orientadores:	Roberto Juncker
Prof. Dr Edson C. Kitani	autotronicarj@gmail.com
Prof. Weslley M. Torres	18/01/2015 19:24:17 — 1/3

Interface BR 2014

ANALOG IMPUT

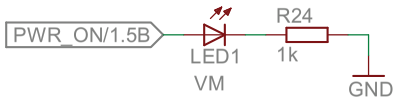


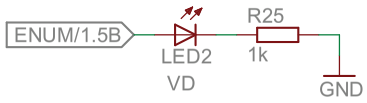
FATEC - Tecnologia em Eletrônica Automotiva	
Interface_BR/USB/LabVIEW	Bruno Zani Sampaio
Orientadores:	Roberto Juncker
Prof. Dr Edson C. Kitani	autotronicarj@gmail.com
Prof. Weslley M. Torres	18/01/2015 19:24:17 — 2/3

Interface BR 2014

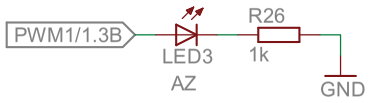
USB, PWM, DIGITAL IMPUT AND OUTPUT

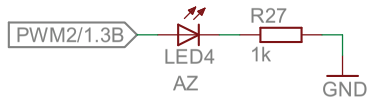
USB CONNECT



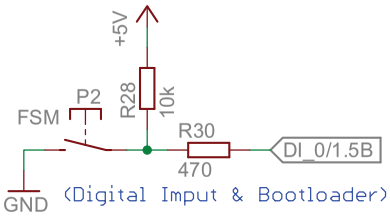


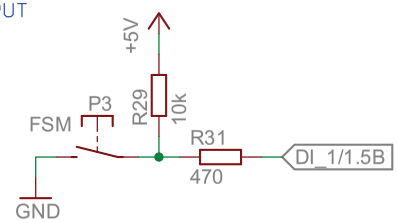
PWM OUTPUT

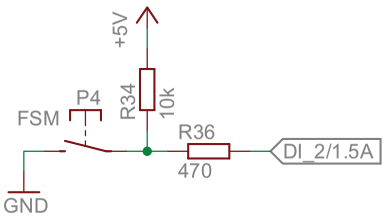


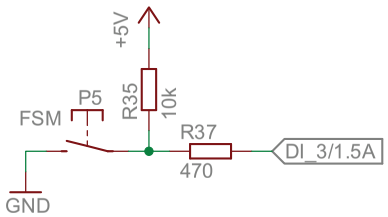


DIGITAL IMPUT

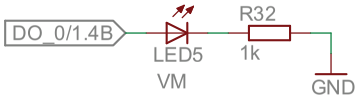


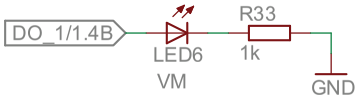


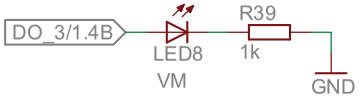




DIGITAL OUTPUT







FATEC - Tecnologia em Eletrônica Automotiva	
Interface_BR/USB/LabVIEW	Bruno Zani Sampaio
Orientadores:	Roberto Juncker
Prof. Dr Edson C. Kitani	autotronicarj@gmail.com
Prof. Weslley M. Torres	18/01/2015 19:24:17 — 3/3

8.2 Lista de componentes

Componentes Semicondutores

Qtd.	Descrição	Valor/ Custo	Componente
1	Microcontrolador	PIC18F4550_40/ R$23,00	IC1
1	*Display* LCD-16X2	LCD-16X2/ 5 V/ R$27,00	LCD1
8	LED 3MM	LED alto brilho/ R$0,80	LED1, LED2, LED3, LED4, LED5, LED6, LED7, LED8

Componentes Passivos

Qtd.	Descrição	Valor	Componente
16	Resistor	470 Ω/ 250 mW/ R$1,60	R2, R3, R4, R5, R9, R11, R13, R15, R17, R19, R21, R23, R30, R31, R36, R37
8	Resistor	1k Ω/ 250 mW/ R$0,80	R24, R25, R26, R27, R32, R33, R38, R39
5	Resistor	10k Ω/ 250 mW/ R$0,50	R1, R28, R29, R34, R35
1	Resistor	330 Ω/ 250 mW/ R$0,10	R6
8	*Trimpot*	2k Ω/ 250 mW/ R$2,40	R8, R10, R12, R14, R16, R18, R20, R22
1	*Trimpot*	10k Ω/ 250 mW/ R$0,30	R7
1	Capacitor Eletrolítico	10 uF/ 16 V/ R$1,50	C7
3	Capacitor Cerâmico	100 nF/ 16 V/ R$0,30	C1, C2, C3
2	Capacitor Cerâmico	22 pF/ 16 V/ R$0,20	C4, C5
1	Capacitor Cerâmico	470 nF/ 16 V/ R$0,10	C6
1	Cristal Oscilador HC49/S	20 MHz/ R$0,80	X1

Componentes Eletromecânicos

Qtd.	Descrição	Valor	Componente
1	Matriz de Contatos	R$62.00	
5	*Pushbutton*	Contato NA/ R$1,25	P1, P2, P3, P4, P5
1	Conector USB	Conector Mini B/ R$1,40	USB
1	Conector MA06-1	Conector ICSP/ R$0,30	SV1

Diversos: Cabo USB para conexão com o PC e jumpers de interligação dos componentes na matriz de contatos.

Custo total (março de 2014): R$ 124,35 (Cento e vinte e quatro Reais).

Fornecedores consultados: www.pontodaeletronica.com.br, www.dabicomercio.com.br, www.compomil.com.br, www.eletrodex.com.br, www.milcomp.com.br, www.cinestec.com.br, www.opoen.com.br.

8.3 *Firmware* principal - Interface_BR

```
/*********************************************************************

                        Interface BR 2014

*********************************************************************/
#include "18F4550.h"
#device ADC=8
#fuses
HSPLL,MCLR,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,CPUDIV1,VR
EGEN,PUT
#use delay(clock=48000000)
#include "pic18_usb.h"
#define  USB_EP1_TX_SIZE 64
#define  USB_EP1_RX_SIZE 3
#include "usb_desc_scope.h"
#include "usb.c"
/*********************************************************************
 Necessário para uso do bootloader HID Microchip 2013/06/15
 Bootloader disponível no Microchip Library Application (MLA) de mesma data
*********************************************************************/
#build(reset=0x1000, interrupt=0x1008)
#org 0x000, 0xFFF { }
#use fast_io(b)
//Pinos de controle do Display LCD
#define rs pin_d0
#define rw pin_d1
#define en pin_d2
```

```
int8  dataRX[2];
int8  dataTX[10];
int8  count=0;
unsigned char data0[]="DAQ Fatec SA";
unsigned int i=0;
void USB_status();
void lcd_ini();
void dis_cmd(unsigned char);
void dis_data(unsigned char);
void lcdcmd(unsigned char);
void lcddata(unsigned char);

void main(void)
{
  port_b_pullups(true); //resistores pullups internos do portb ativados
  setup_port_a(AN0_TO_AN7); //8 entradas analógicas habilitadas
  setup_adc(ADC_CLOCK_INTERNAL); //clok interno para o conversor AD do pic
  setup_timer_0 (RTCC_EXT_H_TO_L); //configuração para borda de descida
  set_timer0(0); //zera a contagem do Timer0
  setup_timer_2(T2_DIV_BY_1, 200, 15); //250us
  set_pwm1_duty(0); //sem duty cicle inicialmente para pwm1
  set_pwm2_duty(0); //sem duty cicle inicialmente para pwm2
  setup_ccp1(CCP_PWM); //CCP1_PWM1_pino RC2
  setup_ccp2(CCP_PWM); //CCP2_PWM2_pino RC1
  set_tris_a(0b11111111); //configuração entradas e saídas do porta
  set_tris_b(0b11110000); //configuração entradas e saídas do portb
```

```
  set_tris_c(0b00000000); //configuração entradas e saídas do portc
  set_tris_d(0b00001000); //configuração entradas e saídas do portd
  output_c(0b00000000); //seta para nível baixo todo o portc
  output_b(0b00000000); //seta para nível baixo todo o portb
  usb_init();//inicializamos a USB
  delay_ms(100);
  USB_status();  //verifica enumeração do dispositivo USB
  lcd_ini(); //inicialização do LCD
  while(data0[i]!='\0')//incrementa a string para que seja plotada letra após letra
  {
        dis_data(data0[i]);
        Delay_ms(200);
        i++;
  }

  while (TRUE)
  {
         usb_task(); //habilita o periférico USB e interrupções
        if(usb_enumerated()) //verifica se o dispositivo foi configurado pelo host
        {
                if (usb_kbhit(1)) //Verifica se o endpoint de saída contém dados
        {
      usb_get_packet(1, dataRX, 3); //recebe-se um pacote de dados com tamanho de 3 bytes e armazena-o na variável "dataRX"
      set_pwm1_duty(dataRX[0]); //o usuário seleciona o duty através do LABView (este valor é armazenado na pos. 0 de "dataRX"
```

```
      set_pwm2_duty(dataRX[1]); //o usuário seleciona o duty2 através do LABView (este valor é armazenado na pos. 1 de "dataRX"
      output_b(dataRX[2]); //envia para o portd (LEDs)
      }
      for(count=0;count<8;count++)   //  laço FOR para seleção dos 8 canais AD
      {
      set_adc_channel(count);
      delay_us(20); //tempo necessário para carregar o capacitor interno do PIC
      dataTX[count]=read_adc(); //os valores das conversões AD são armazenados em 8 bytes (conversão de 0-255 por canal)
      }
      dataTX[8]=get_timer0(); //o 9º byte de "dataTX" é utilzado para armazenar o retorno do valor da contagem do Timer0
      dataTX[9]=input_b(); //o 10º byte de "dataTX" é utilizado para armazenar o estado lógico do portb (push-bottom e LCD)
      usb_put_packet(1, dataTX,64, USB_DTS_TOGGLE); //Envia-se um pacote de dados com tamanho de 10 bytes armazenados na variável"dataTX" para o host
    }
  }
}

void lcd_ini()                    //Inicializa o display LCD no modo 4-bit mode.
{
        dis_cmd(0x02);
        dis_cmd(0x28);
        dis_cmd(0x0C);
        dis_cmd(0x06);
```

```
        dis_cmd(0x80);
}
void dis_cmd(unsigned char cmd_value)
{
        unsigned char cmd_value1;
        cmd_value1 = (cmd_value & 0xF0);
        lcdcmd(cmd_value1);
        cmd_value1 = ((cmd_value<<4) & 0xF0);
        lcdcmd(cmd_value1);
}
void dis_data(unsigned char data_value)
{
        unsigned char data_value1;
        data_value1=(data_value&0xF0);
        lcddata(data_value1);
        data_value1=((data_value<<4)&0xF0);
        lcddata(data_value1);
}

void lcdcmd(unsigned char cmdout)
{
        output_d(cmdout);
        output_low (rs);
        output_low (rw);
        output_high(en);
        Delay_ms(10);
```

```
        output_low (en);
}
void lcddata(unsigned char dataout)
{
        output_d(dataout);
        output_high(rs);
        output_low (rw);
        output_high(en);
        Delay_ms(10);
        output_low (en);
}
void USB_status() //Subrotina que indica a enumeração do dispositivo USB
{
  output_high (PIN_C6); //Acende o led vermelho (erro).
  output_low (PIN_C7); //Apaga o led verde (USB_OK).
  usb_wait_for_enumeration(); //Espera a ser enumerado pelo host.
  output_high (PIN_C7); //Acende o led verde (USB_OK).
  output_low (PIN_C6); //Apaga o led vermelho (erro)
```