UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

TRABALHO DE CONCLUSÃO DE CURSO

BARRAMENTO UNIVERSAL SERIAL - USB

FÁBIO ALCANTARA ROCHA

Campina Grande - PB
Agosto de 2009

FÁBIO ALCANTARA ROCHA

TRABALHO DE CONCLUSÃO DE CURSO

BARRAMENTO UNIVERSAL SERIAL – USB

Relatório apresentado à Coordenação de Graduação em Engenharia Elétrica da UFCG, como parte dos requisitos para obtenção do título de Engenheiro Eletricista.

Aprovado em_____ de _____ de 2009

Banca Examinadora

_____

Orientador: Prof. Edmar Candeia Gurjão

_____

Professor convidado: Bruno Barbosa Albert

AGRADECIMENTOS

Agradeço em primeiro lugar, aos meus pais Aurelino Souza Rocha e Oleniva do Nascimento Alcantara Rocha, a minha esposa Albanisa Araújo Teixeira, meu filho Isaac Henric Araújo Alcantara e minha irmã Patrícia Alcantara Rocha, por terem me dado o apoio que sempre precisei, aconselhando, incentivando e acreditando no meu potencial.

1. **APRESENTAÇÃO**

2. **BARRAMENTO USB (UNIVERSAL SERIAL BUS)**

   2.1. Principais características

   2.2. Topologia do barramento USB

   2.3. Controlador de *HOST*

   2.4. Estrutura elétrica

3. **TÉCNICAS DE CONTROLE DE TRÁFEGO E DADOS**

   3.1. Tipos de Transferência de dados

   3.2. Protocolo de comunicação

      3.2.1. Descrição dos pacotes

      3.2.2. *Endpoint*s e *Pipes*

   3.3. Descritores de Identificação

4. **PROCESSO DE ENUMERAÇÃO**

5. **ATIVIDADES DESENVOLVIDAS**

6. **CONCLUSÕES**

7. **BIBLIOGRAFIAS**

1. APRESENTAÇÃO

O Objetivo deste trabalho consiste em realizar um estudo da interface de comunicação *Universal Serial Bus* – USB, técnicas de controle de tráfego de dados através do barramento e algumas API's (*Application Programming Interface)* necessárias para desenvolvimento de aplicações que permitirão conexões com periféricos externos ao computador (*pendriver*, mouse, teclados, no-break, câmeras, impressoras, entre outros).

2. BARRAMENTO USB (*UNIVERSAL SERIAL BUS*)

2.1 Principais características

O barramento USB é uma entrada de dados que permite a utilização de uma entrada única para conectar vários periféricos externos ao dispositivo onde essa entrada está instalada. Em uma mesma entrada USB pode-se conectar até 127 dispositivos no barramento , sendo está, uma das principais diferenças em relação às tradicionais portas seriais e paralelas dos computadores, as quais permitem que apenas um periférico seja conectado por porta. Este tipo de tecnologia está presente em muitos dispositivos disponíveis no mercado como**:** no-break**,** aparelhos de som, relógios, scanners, MP3 players, impressoras, webcams, entre outros.

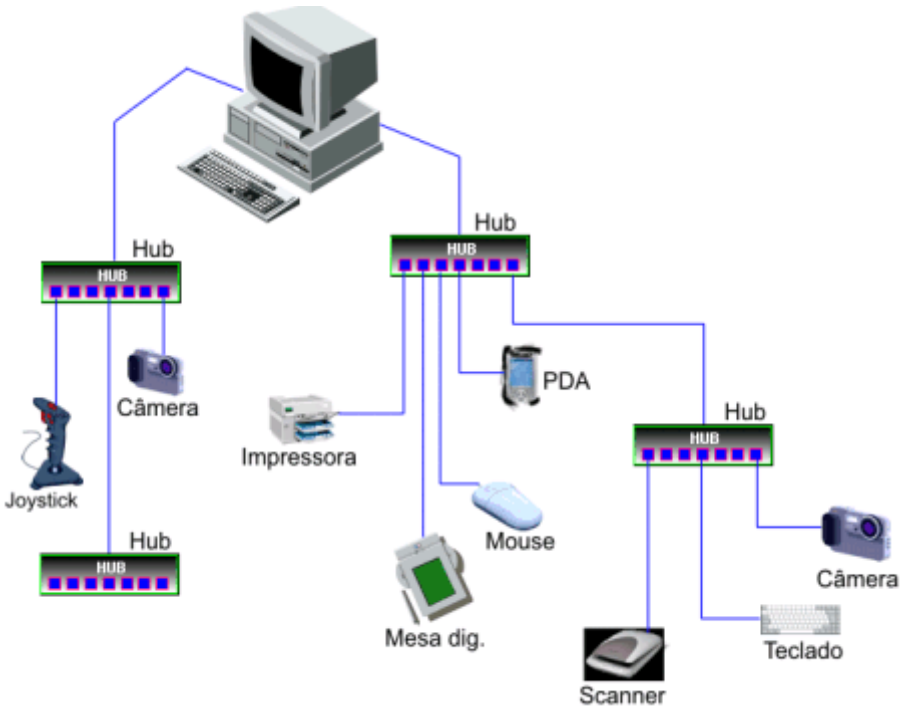

Figura 01- Barramento USB porta padrão para vários periféricos

http://www.rogercom.com/PortaUSB/ArvoreUSB.gif

O barramento USB acaba de vez com inúmeros problemas de falta de padronização do PC moderno e diversos dispositivos no mercado. Uma das grandes vantagens do USB é que o próprio usuário pode instalar um novo periférico, sem a menor possibilidade de gerar algum conflito ou, então, queimar alguma placa.

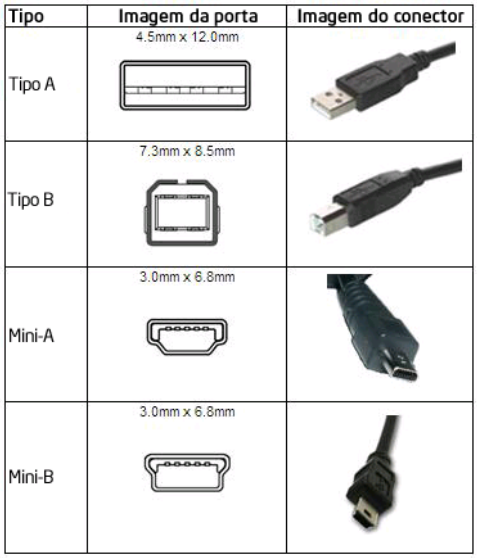

Figura 02 – Conectores USB

http://www.intel.com/support/pt/motherboards/desktop/sb/img/usb.jpg

O conector USB (Fig. 02) é padronizado. Em periféricos pequenos, encontraremos somente um conector USB, a ser conectado a uma tomada USB. Em periféricos maiores (como teclados e impressoras), temos, além do conector USB, alguns terminais USB para a conexão de periféricos, portanto fazendo o "cascateamento" do barramento. Além disso, novas conexões podem ser conseguidas com a instalação de *hub*'s USB ("concentradores"), periféricos que expandem a quantidade de portas do barramento.

O barramento USB é *plug-and-play,* ou seja, pode-se encaixar e desencaixar periféricos com o micro ligado, e o sistema operacional

automaticamente detecta que um novo periférico USB foi adicionado (*hotplug*). Isso e possível graças ao controlador USB presente na placa-mãe e que, em geral, esta integrada no *chipset* (conjunto de circuitos integrados). Caso o sistema operacional necessite de um *driver* para um periférico recém-instalado (um *pendriver*, uma impressora ou um tocador de mp3, por exemplo), pedirá ao usuário que insira o disco contendo o *driver* necessário.

Devido a uma cooperação entre Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC e Philips, uma das primeiras versões do barramento foi a USB 1.0 com velocidade de 1,5Mbs (*low speed*) sucessivamente a 1.1 com velocidade de 1,5Mbps a 12Mbps. Mas tarde surgiria a versão 2.0 (*high speed*) com velocidade para transferência da informação da ordem de 480Mbs, e mais atualmente (novembro de 2008) a USB 3.0 á qual pode transferir 27GBps.

A fim de permitir a conexão máxima de até 127 dispositivos em uma única porta, os *Hubs* USB são elementos de extremamente importância na topologia de uma distribuição USB, os mesmos fornecem meios físicos que possibilitam inserção de novos dispositivos.

Figura 03 – Hub USB

http://tbn2.google.com/images?q=tbn:H4kiODrm5-rhvM:http

Os *Hub's* USB podem ser divididos em duas configurações com relação à alimentação elétrica:

- ***Hubs sem fonte de alimentação (Bus-powered)***: Possuem até quatro portas *downstream* (portas secundárias que servem para conectar outro hub ou dispositivos), fornecendo cada uma 100mA e alimentam-se da própria corrente do barramento USB.
- ***Hubs com fonte de alimentação (Self-powered)***: Esses *hubs* podem fornecer da porta um corrente de até 500mA.

Vale salientar que um dispositivo USB, ao estar conectado em uma das portas de conexão do *hub*, se uma corrente superior a capacidade suportada pelo *hub*, o dispositivo entrará em um estado de conexão física mas não lógica ou seja não se comunica.

2.2 Topologia USB

A topologia física do padrão USB é composta por um Controlador de *HOST* e um *Hub* Raiz, onde através das portas do *Hub* Raiz conectamos dispositivos e *Hub*'s para formarmos uma rede USB (Ver Fig. 04).

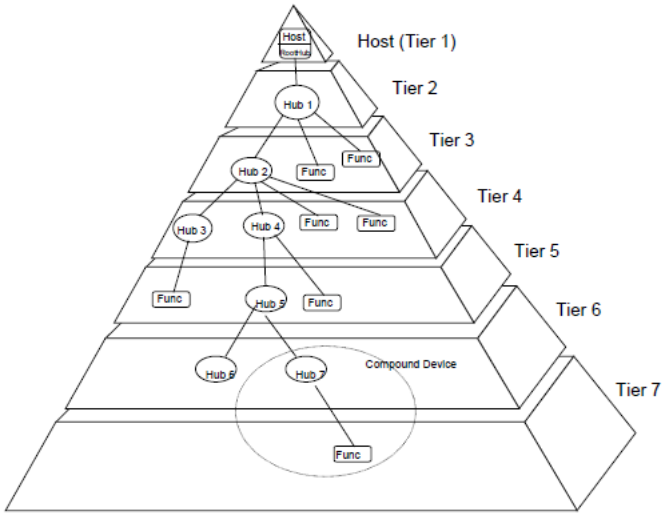

Figura 04 – Topologia USB

Universal Serial Bus Specification Revision 2.0 – pag.16

Vale salientar que a estrutura apresentada na Fig. 04 pode variar o nível de empilhamento da pirâmide**,** por exemplo, a USB 1.1 pode estabelecer até 4 *hub*, já para os USB 2.0 é possível enfileirar até 6 *hub*. Nessa estrutura é importante ressaltar que o comprimento máximo entre um *hub* e um dispositivo é cerca de 5 (cinco) metros.

2.3 Controlador de *HOST*

Como o próprio nome sugere o controlador de *HOST* tem como objetivo controlar o fluxo de dados entre os dispositivos conectados, monitorar os sinais do bus USB e detectar a entrada e retirada de dispositivos.

Devido ao consórcio das empresas que desenvolveram o padrão USB, chegou-se aos seguintes modelos de controladores:

- **UHCI (*Universal HOST Controller Interface*)**
- **OHCI (*Open HOST Controller Interface*)**

- **EHCI (*Enhanced HOST Controller Interface*)**

Os Controladores de *HOST* combinam suas funções de controle entre um *driver* controlador de *HOST* (HCD) e um Controlador *HOST*. Característica bastante presente na versão USB 2.0 onde o *driver* e o hardware, trabalham dividindo o serviço para que os dispositivos principalmente de áudio e vídeo funcionem em alta velocidade. Na figura 05 poderemos visualizar estrutura do controlador de *HOST*.

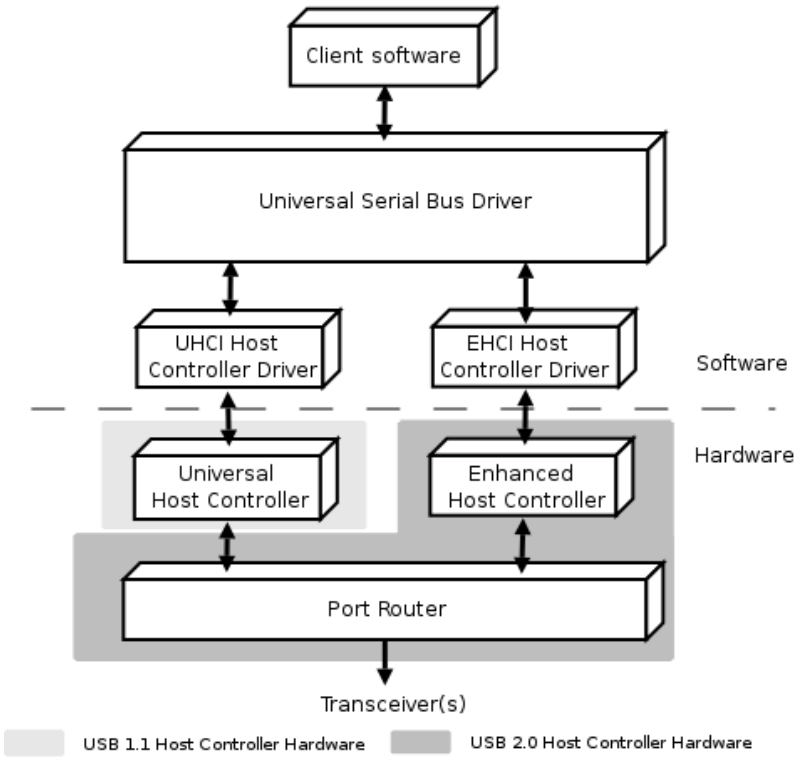

Figura 05 – Controlador de HOST visão geral
http://gaisler.com/images/grusbhc-HOST.png

2.4 Estrutura Elétrica

O cabo do barramento USB é composto pela seguinte configuração:

- Uma malha para eliminação de ruídos simples;

- 4 fios (VBus, D+, D- e GND) onde o fio vermelho é o positivo, o preto é o gnd (terra) e os outros dois D+ e D- são usados pelo sistema USB para transferência de dados entre os dispositivos.

Os sinais que trafegam entre os dois fios D+ e D- usam uma codificação NRZI (*no return to zero invert*). O bit 1 é codificado através de uma transição ocorrendo da maior voltagem para a menor, ou também o inverso, da menor para a maior após sinalização do relógio. Já o bit 0 é codificado sem haver transição. Durante o intervalo de um bit a voltagem é constante.

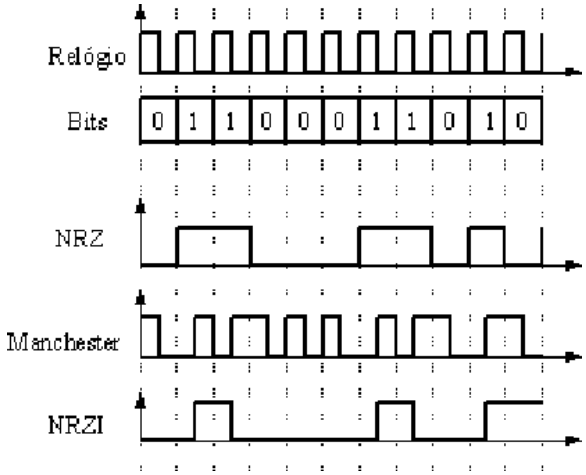

Figura 06 – Codificação NRZI

http://images.google.com.br/imgres?imgurl=http://penta

A interface USB disponibiliza uma maneira de detectar se os dispositivos conectados no barramento são de alta ou baixa velocidade, bem como, detectar sua presença. Para isso uma configuração física entre o *HOST* e o dispositivo (Fig. 07) com um resistor de Pull-up de 1,5 k ohm ligado desde o sinal D+ a uma tensão de 3,3V. Este resistor serve para avisar o controlador *HOST* que um novo dispositivo foi conectado ao barramento USB e também informar que o dispositivo irá se comunicar com o *HOST* em alta velocidade. Para dispositivos

que se comunicam em baixa velocidade o resistor de *pull-up* deve ser conectado ao sinal D-.

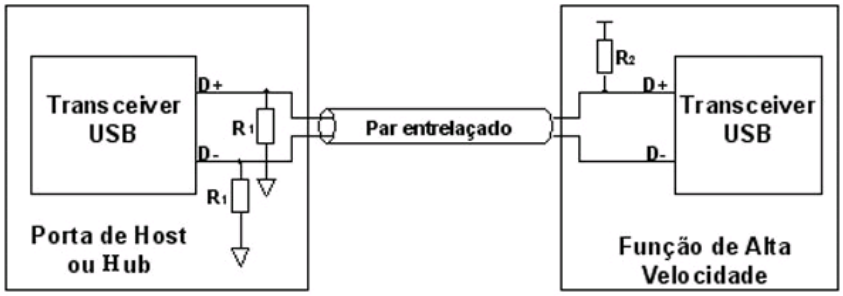

Figura 06 – Esquema físico de conexão do barramento USB
http://www.clubedohardware.com.br/imageview.php?image=1

3. TÉCNICAS DE CONTROLE DE TRÁFEGO E DADOS

3.1 Tipos de Transferência de Dados

O controle do tráfego no barramento USB é realizado exclusivamente pelo dispositivo *HOST*, que para acessar dados através do barramento USB, o processo deve ser inicializado pelo *HOST* quando o mesmo recebe uma respectiva solicitação da aplicação que deseja enviar ou receber dados para o periférico externo (*nobreak*, câmeras, teclados entre outros). A arquitetura USB compreende quatro tipos básicos de transferências de dados:

- ***Controle:*** A transferência de dados de controle é utilizada pelo *HOST* para configurar um dispositivo e requisitar informações sobre o seu estado. Essa transferência é usada para configurar um dispositivo no instante de sua conexão.

- ***Interrupção*:** Esse tipo de transferência é utilizado por dispositivos que usam pouca banda do sistema ou que precisam enviar e receber dados de maneira não freqüente, mas com certos limites de tempo para execução.

- ***Isossíncrona:*** Tipo de transferência composta somente de dados, ideal para dispositivos que não necessitam checar se os dados chegaram corretamente ao destino. Não possui detecção de erro e nem utiliza CRC (*Cyckic Redundancy Check*). Esse tipo de transferência é mais utilizado por dispositivos de vídeos, áudios e outros que precisam de uma velocidade constante de transmissão, não importando a perda de dados por erros ocasionais. A transferência isossíncronas tem prioridades em termo de largura de banda.

- ***Bulk:*** A transferência de grandes volumes de dados como aquisição de dados de dispositivos tipos: impressoras, *pendriver*, câmera entre outros. Nesse tipo de transferência há detecção e correção de erros, confirmação e recuperação de pacotes de dados corrompidos.

  Nesse tipo de transferência a largura de banda pode variar se houver outras atividades concorrentes sendo executadas no barramento USB. Esse tipo de transmissão garante a segurança no tráfego de dados, mas não garante velocidade constante em que os mesmos são transmitidos.

3.2 Protocolo de Comunicação USB

3.2.1 Pacotes de transferência de dados

O protocolo de aplicação do barramento USB é baseado no envio de pacotes, onde o *HOST* é quem inicia todas as transações. Para isto, a USB define três tipos diferentes de pacotes: Pacotes *Token* (*token packets*), Pacotes de Dados (Data *Packets*) e Pacotes *Handshake* (*Handshake Packets)*. Ver figura 07.

Uma vez o dispositivo USB conectado no *HOST*, o primeiro pacote gerado é o *Token* para descrever se transação de dados entre os dispositivos será de escrita ou leitura. O pacote que se segue é um pacote de dados, seguido por um pacote de *Handshake*, o qual informará o sucesso na transação da comunicação.

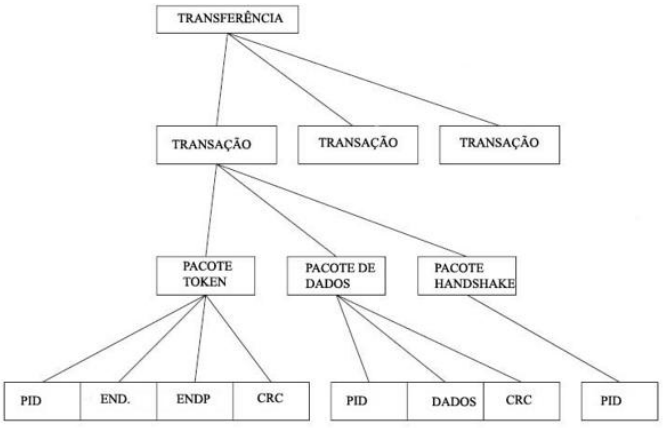

Figura 07 – Pacotes de Transferência

http://www.gta.ufrj.br/grad/07_1/wusb/TransportedeDados_cli

3.2.2 Descrição dos campos dos pacotes

- **SYNC (*synchronization*):** Este campo tem um tamanho de 8 bits para velocidades baixas e, 32 bits para altas velocidades. E utilizado para

sincronizar o relógio (clock) do transmissor com o do receptor. Todos os pacotes começam com este campo

- **PID (***Packet Identifier***)**: Representa o pacote de identificação, este campo tem um tamanho de 8 bits, e usado para identificar o tipo de pacote que será enviado.

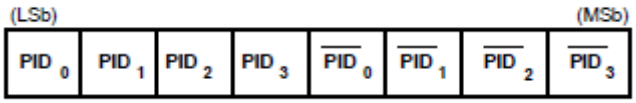

Figura 08 – Formato do campo PID

Universal Serial Bus Specification Revision 2.0 – pag.195

Os bits (PID0 a PID1) indicam o tipo de pacote que pode ser: *Token*, *Data*, *Handshake* e *Special*. Os bits (PID2 e PID3) acrescentam mais informações sobre o tipo de pacote. Os bits (PID_0 a PID_3) são usados para verificação dos dados (cálculos para verificação e prevenção de erros).

- **ADDR (***Address Field***)**: Este campo define o endereço de destino do dispositivo ao qual o pacote de dados será enviado. Tem um tamanho de 7 bits, podendo endereçar no Máximo ate 127 dispositivos distintos, conectados ao barramento UBS.
- **ENDP** (*Endpoint Field*): Este campo é conhecido como *Endpoint* e tem o tamanho de 4 bits, permitindo 16 possíveis *Endpoints* diferentes. O *Endpoint* de número 0 (zero) é usado pelo *HOST* para se comunicar com todos os dispositivos conectados ao bus USB, para configurá-los ou obter informações.

- **DATA**: São os dados para transferência, este campo pode armazenar ate 1024 bytes.
- **CRC (**Cyclic Redundancy Check): O objetivo deste campo é armazenar o resultado de um cálculo em nível de bits executados em alguns campos do pacote, para garantir a segurança dos dados transmitidos. Ou seja, se os dados se alterarem durante a transmissão, o receptor poderá comparar o valor deste campo, calculando o CRC do pacote recebido e correção do erro se houver. O campo PID não e agregado a esse calculo porque tem seu próprio recurso de correção de erros. Para os pacotes *Token* esse campo tem o tamanho de 5 bits (CRC5), e para pacotes de dados, tem tamanho de 16 bits(CRC16).
- **EOP:** Este campo indica o fim de qualquer pacote o mesmo é representado com um simples 0 (zero) por aproximadamente 2 a 3 bits de tempo.

3.2.3 *Endpoint* e Pipes

Endpoint (ponto final de um dispositivo) Área de memória (*Buffer*) reservado no dispositivo para armazenar os dados que trafegam em um tubo (*pipe*).

Um dispositivo USB pode ter no máximo 16 *Endpoins* na versão 2.0 do USB. O *Endpoint* 0 (zero) e reservado para o *HOST* obter informações sobre cada dispositivo conectado ao barramento USB. Através deste *Endpoint*, o *HOST* pode enviar comandos de controle em baixa velocidade para obter informações descritivas sobre o dispositivo, como: número de série, fabricante, classe, subclasse, versão do barramento USB, nome do produto, tipo do protocolo, números de *Endpoint*s, interfaces, velocidade, tipo de transferência, entre outras.

Após a aquisição de todas estas informações sobre o dispositivo, o *HOST* estabelece realmente uma comunicação com a aplicação.

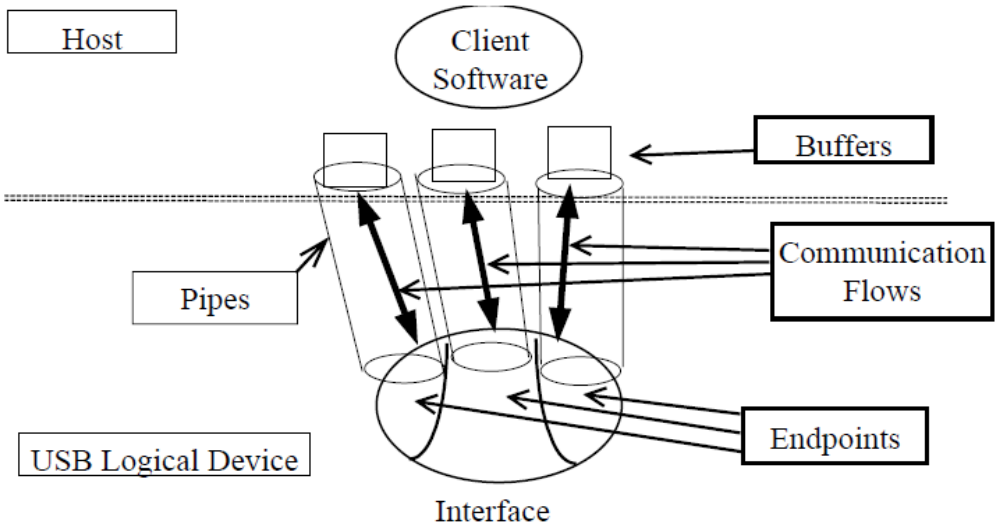

Figura 09 – Fluxo de comunicação USB

Universal Serial Bus Specification Revision 2.0 – pag.33

Pipe (tubo) é uma associação entre um *Endpoint* no dispositivo e um software no *HOST* (ver Fig. 09). *Pipe* não é algo físico, é uma via de comunicação virtual que só existe em um nível de software. Quando um dispositivo USB se conecta ao sistema, o *HOST* cria uma configuração para se comunicar com o dispositivo. Um Pipe pode ser descrito também como uma via de comunicação unidirecional ou bidirecional, onde podem existir *Endpoint*s de entrada (*In*) e saída (*out*). Os *Pipes* passam a existir quando um dispositivo é configurado pelo *HOST*. Enquanto um dispositivo estiver sendo alimentado e conectado ao controlador *HOST*, sempre existirá um Pipe de controle padrão (*Control Default*) para que esse dispositivo

possa fornecer acesso às configurações, como, informações de controle e estado do mesmo.

Com o objetivo de estabelecer vias de comunicação com os *Endpoint*, o barramento USB define dois tipos de *Pipes*:

- **Stream**: É uma via de comunicação unidirecional entre um software no *HOST* e um *Endpoint*, que pode ser dos tipos: Interrupção, Isossíncrono, ou *Bulk*. Se um dispositivo precisar transferir de forma bidirecional um dos tipos de *Endpoint*, o sistema deve estabelecer dois Pipes: um definido com *Endpoint* de saída (*out*), e outro com *Endpoint* de entrada (*In*).
- **Mensagem ou Controle:** É uma via de comunicação bidirecional entre um software no *HOST* e dois *Endpoint*s de Controle em um dispositivo USB. Ou seja, é um tubo de dois *Endpoint*s: um de entrada (In) e outro de saída (out) que tem uma estrutura de dados bem definida. É através de um *Pipe* deste que o *HOST* usa o *Endpoint* 0, para obter os descritivos dos dispositivos para configurá-los no sistema USB.

3.3 Descritores de Identificação

Todos os dispositivos USB têm uma hierarquia de descritores que informam ao *HOST* o que o dispositivo é, ou seja, sua “identidade”, suas características de funcionamento, como; número de série do produto, identificação do fabricante, tipo do dispositivo (no-break, *pendriver*, teclados, câmeras, scanner, modem, mouse, etc.), numero de configurações, número de *Endpoint*, tipo de transferência, tipo de interface, etc.

O sistema USB define padrões de descritores para dispositivos, descritores de configuração, descritores de interface, descritores de *Endpoint*s e descritores de classe.

- **Descritor de Dispositivo:** Informa para o *HOST* saber qual a versão do USB, que o dispositivo suporta. Alguns dos atributos desse descritor são: identificação do fabricante, número de serie, ID do Vendedor e Produto, classe e subclasse do dispositivo.
- **Descritores de Configuração:** Carregam informações sobre a capacidade e funcionalidade do dispositivo, como tipo de alimentação de energia (*Bus powered* ou *Self powered*), máxima corrente consumida pelo dispositivo, ente outras.
- **Descritores de Interface:** Informa sobre o número de *Endpoint* que suporta o protocolo utilizado, e algumas strings de texto especificando o nome do produto.
- **Descritores de *Endpoint*:** A USB sempre assume que o *Endpoint* 0 é um um *Endpoint* de controle e, é configurado antes de qualquer descritor. Fora o *Endpoint* 0, cada *Endpoint* deve ser configurado, o quê consiste em definir o número do *Endpoint*, direção da comunicação (*In*, *Out*) e tamanho do pacote de dados a transmitir.
- **Descritor de Classe:** Determina a classe do dispositivo, ou seja, quando cada dispositivo com conexão USB (impressoras, pendriver, dispositivos de comunicação, mouses, teclados e etc) ele é identificado como sendo de uma classe definida pelo padrão 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
DCh	Both	Diagnostic Device
E0h	Interface	Wireless Controller
EFh	Both	Miscellaneous
FEh	Interface	Application Specific
FFh	Both	Vendor Specific

Tabela 1 – Classes de Dispositivos

http://www.usb.org/developers/defined_class

4. PROCESSO DE ENUMERAÇÃO

O processo de enumeração consiste na conexão, detecção, leitura dos descritivos dos dispositivos e desconexão. Este processo é controlado em tempo real pelo controlador de *HOST* e o software do sistema. Do ponto de vista do computador, o processo de enumeração é transparente, desde que se tenha antes instalado no sistema operacional, os *drives* do fabricante do dispositivo. Após isso, o carregamento dos *drives* quando um dispositivo é conectado, é automático.

O *HOST* inicia o processo de enumeração, pelo envio de um pedido padrão ao *Endpoint* 0 do dispositivo, desta forma para que o processo de enumeração seja

bem sucedido, o dispositivo deve responder aos pedidos do *HOST*. Abaixo elencamos um passa a passo para processo de enumeração.

1. O dispositivo é conectado ao *HOST*;
2. O controlador *HOST* detecta o dispositivo através de seu sinal elétrico;
3. O *HOST* provoca uma interrupção de reset para que o dispositivo reinicie limpando as variáveis de registros para que se possa iniciar a enumeração;
4. O *HOST* define um canal de comunicação com o dispositivo;
5. O *HOST* atribui um único endereço ao dispositivo;
6. O *HOST* requisita os descritores do dispositivo, identifica-os, carrega o driver adequado que irão interagir com a aplicação.

Uma vez que a seqüência citada tenha sido completada, é estabelecida a fase de gerenciamento do dispositivo e das comunicações de dados. Após isso, o dispositivo estará pronto para transferir ou receber dados.

5. ATIVIDADES DESENVOLVIDAS

Com o objetivo de familiarizar-se com as nuances do barramento USB, inicialmente utilizamos o programa USB View, para identificarmos através desta ferramenta, o processo de identificação dos dispositivos conectados ao barramento USB, bem como todos os descritores.

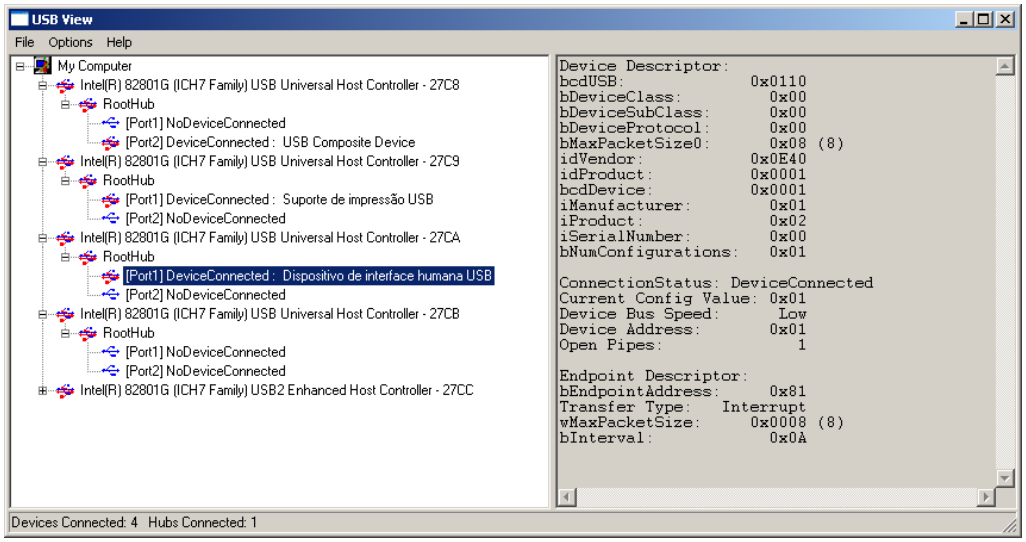

Figura 10 – USB View

Na tentativa de estudar algumas API's (*Application Programming Interface*) necessárias para desenvolvimento de aplicações capaz de manipular dados através do barramento USB. Concentramos nossos estudos em um exemplo do livro "USB Design by Example" [1], o qual exemplifica através de código (linguagem programação C) todo o processo de identificação dos dispositivos conectados ao barramento USB.

O compilador utilizado para desenvolvimento da aplicação exemplo foi o *Microsoft Visual C++ 6.0.*

```
// Programa principal
int main(int argc, char* argv[])
{
	HANDLE Pc_HOST;
	char HOST_name[]="\\\\.\\HCD0";
	int i;
	DWORD erro=0;
	FILE *stream;

   DEBUG = true; //Faz com que a saída seja apresentada com mais (DEBUG = true) ou com
		 //menos (DEBUG = !true) detalhes

	printf("USB Design:\n");

	printf("Re -  Directing console output to 'Display USB.txt.\n\n");
	stream = freopen("DisplauUSB.txt","w",stdout);
	if( stream == NULL )
		printf("Cannot redirect output\n\n");

	for(i=0;i<10;i++)
	{
		HOST_name[7] = i + '0';// Muda o último caractere da string previamente
				// armazenada em HosContrtollerName, fazendo com
				// que esse caractere seja 0, 1, 3 ... 9.

	Pc_HOST = CreateFile(HOST_name,
				GENERIC_WRITE,
					FILE_SHARE_WRITE,
					&SA,
					OPEN_EXISTING,
					0,
					NULL
					);

		if(Pc_HOST != INVALID_HANDLE_VALUE) printf("HOST controller handle %d
created\n", HOST_name);

		if( (DEBUG) & (erro==0))
		{
			printf("\nControlador de HOST %s encontrado.",HOST_name);
			erro = EnumerateHOSTController(Pc_HOST); //Inicia o processo de
enumeração
			CloseHandle(Pc_HOST);
		}
	}
	fclose(stream);
	system("notepad DisplauUSB.txt");

	return erro;
}
```

A função “*EnumerateHOSTController (Pc_HOST)”* da início ao processo de enumeração fazendo com que o *HOST* solicite um pedido (*Endpoint* 0) aos dispositivos conectado no barramento USB.

Uma vez que o processo de enumeração foi consolidado o que obteremos como resposta será:

```
HOST controller handle 1245040 created

Controlador de HOST \\.\HCD0 encontrado.
Nome do sistema eh {36FC9E60-C465-11CF-8056-444553540000}\0004
E o nome do hub ah USB#ROOT_HUB#4&e097488&0#{f18a0e88-c30c-11d0-8815-00a0c906bed8}
Root hub handle 40 created
Em GetPortData com HubHandle = 28, PortCount = 1, HubDepth = 0
                Port[1] = No device connected
Closing root hub handle 40
HOST controller handle 1245040 created

Controlador de HOST \\.\HCD1 encontrado.
Nome do sistema eh {36FC9E60-C465-11CF-8056-444553540000}\0003
E o nome do hub ah USB#ROOT_HUB#4&765d3eb&0#{f18a0e88-c30c-11d0-8815-00a0c906bed8}
Root hub handle 40 created
Em GetPortData com HubHandle = 28, PortCount = 1, HubDepth = 0
                Port[1] = I/O device connected
In DisplayDeviceDescriptor with HubHandle = 28, PortIndex = 1, LanguageID = 0
*****************************
*  ==> Device Descriptor<== *
*****************************
 Comprimento Buffer 12
 Tipo do descritor 01
 bcdUSB 0110
 bDeviceClass 00
 bDeviceSubClass 00
 bDeviceProtocol 00
 bMaxEP0Size 08
 wVendorID 04b8
 wProductID 0005
 wDeviceID 0100
 iManufacturer <<01>>
.
Continua…….
```

Fazendo um paralelo com os dados obtidos com USB View podemos notar que as informações dos descritores são as mesmas. Mas com a diferença que o

procedimento para obter esses parâmetros é mais transparente para o desenvolvedor.

6. CONCLUSÃO

A realização deste trabalho de conclusão de curso – TCC, possibilitou um maior conhecimento sobre interfaces de comunicação, em especial o barramento USB, além de proporcionar um maior contato com ferramentas de desenvolvimento e análise (compiladores e API's do Windows) para aplicações que necessitem utilizar a conexão USB como meio para transferência de informação.

7.0 Bibliografia

[1] Usb Design By Example. Hyde, John, JOHN WILEY CONSUMER, 1999.

[2] Usb Complete. Axelson, Jan, LAKEVIEW RESEARCH, 3ª Ed.,2005.

[3] Usb hardware and software,Jaff, kosar, ANNABOOKS, 1ª Ed., 1998.

[4] Universal Serial Bus Specification, Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips, Revision 2.0 April 27, 2000.

[5] Curso USB/Serial Controle de Dispositivos, Rogério, Antônio, 1ª Ed. 2007.

[6] www.usb.org Último acesso efetuado em: 10/08/2009.

[7] www.alanmacek.com Último acesso efetuado em: 08/08/2009.

[8] www.lvr.com Último acesso efetuado em: 10/08/2009.

[9] www.usb-by-example.com Último acesso efetuado em: 13/08/2009.