Guia Rápido de Comunicação USB via HID Terminal em Microcontroladores PIC

Autor: Eng. Rahul Martim Juliato (rahul.juliato@gmail.com)

Este artigo apresenta de forma prática alguns passos para realizar uma comunicação enviando e recebendo dados de um microcontrolador PIC 18F4550 pela interface USB. O compilador utilizado foi o MIkroC Pro 5.3 e o simulador foi o Proteus Isis 7.7.

Para configurar o PIC como um USB HID Device, é necessário gerar um arquivo de descrição de funções de acordo com os padrões USB HID encontrados em http://www.usb.org/developers/hidpage/. Nessa página há um programa chamado "Hid Descriptor Tool" para gerar suas próprias configurações de dispositivos. É possível gerar teclados, mouses, game controllers entre outros, esse artigo foca em comunicação padrão via terminal e não aprofunda esses tópicos.

Sumário

Programa 1

Simulação 9

Conclusão 14

Referências e Agradecimentos 14

Programa

Após criar um novo projeto no MikroC Pro, deve-se abrir a ferramenta "HID Terminal" que se encontra no menu Tools.

Na aba Descriptor deve-se preencher os dados de configuração do arquivo a ser gerado conforme mostra a figura 1, em seguida deve-se clicar em "Save Descriptor" e salvar conforme a figura 2.

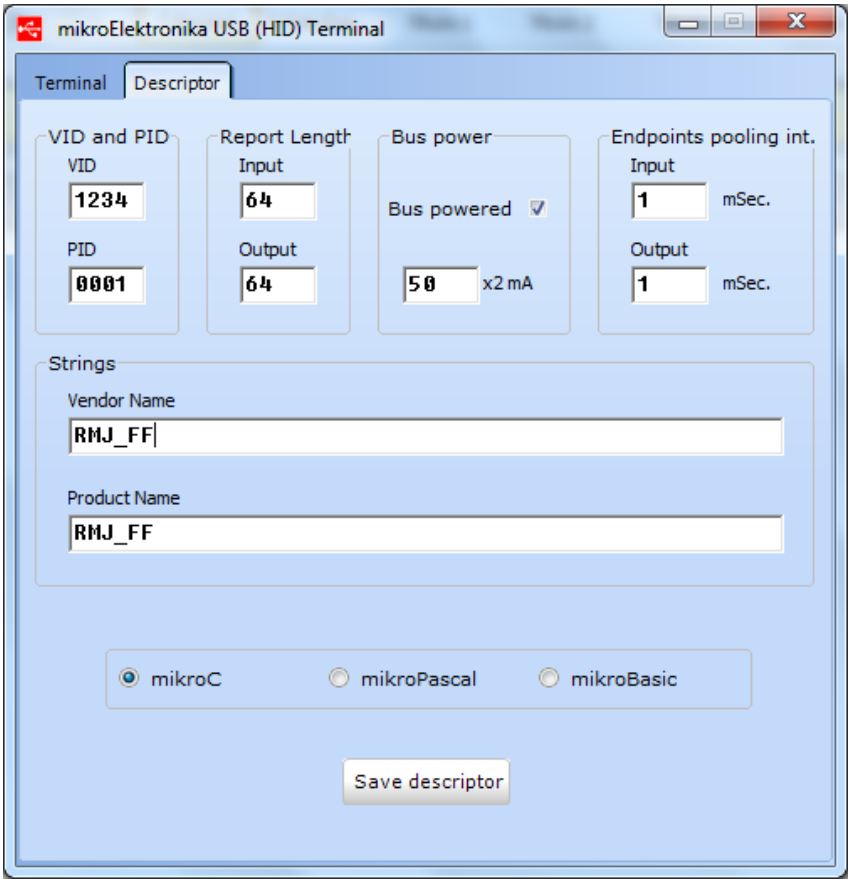

Figura 1 – Configuração do Arquivo Descriptor

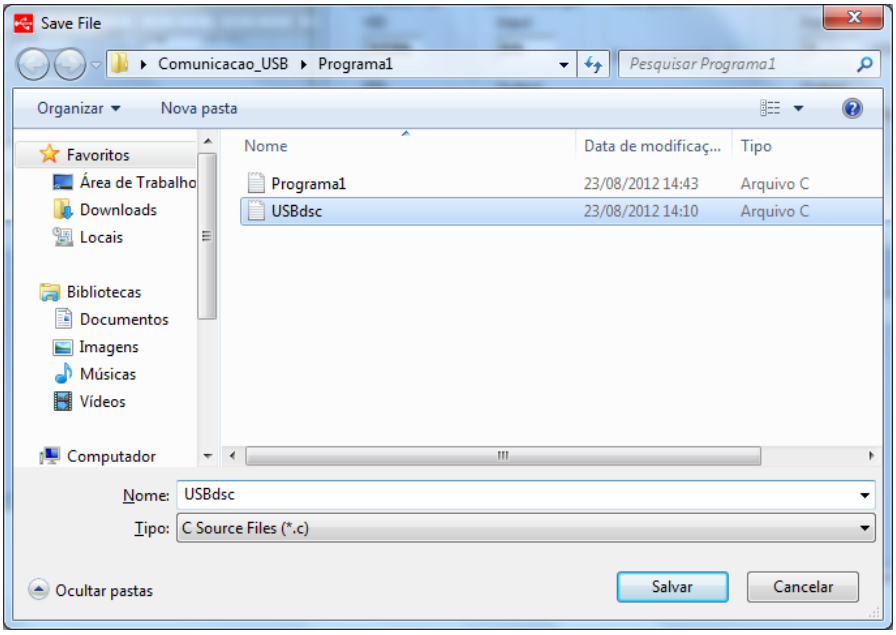

Figura 2 – Salvando um Arquivo Descriptor

Código gerado pela ferramenta e salvo em USBdsc.c:

```
const unsigned int USB_VENDOR_ID = 0x1234;
const unsigned int USB_PRODUCT_ID = 0x0001;
const char USB_SELF_POWER = 0x80;            // Self powered 0xC0,  0x80 bus powered
const char USB_MAX_POWER = 50;               // Bus power required in units of 2 mA
const char HID_INPUT_REPORT_BYTES = 64;
const char HID_OUTPUT_REPORT_BYTES = 64;
const char USB_TRANSFER_TYPE = 0x03;         //0x03 Interrupt
const char EP_IN_INTERVAL = 1;
const char EP_OUT_INTERVAL = 1;

const char USB_INTERRUPT = 1;
const char USB_HID_EP = 1;
const char USB_HID_RPT_SIZE = 33;

/* Device Descriptor */
const struct {
    char bLength;               // bLength         - Descriptor size in bytes (12h)
    char bDescriptorType;       // bDescriptorType - The constant DEVICE (01h)
    unsigned int bcdUSB;        // bcdUSB          - USB specification release number
(BCD)
    char bDeviceClass;          // bDeviceClass    - Class Code
    char bDeviceSubClass;       // bDeviceSubClass - Subclass code
    char bDeviceProtocol;       // bDeviceProtocol - Protocol code
    char bMaxPacketSize0;       // bMaxPacketSize0 - Maximum packet size for endpoint 0
    unsigned int idVendor;      // idVendor        - Vendor ID
    unsigned int idProduct;     // idProduct       - Product ID
    unsigned int bcdDevice;     // bcdDevice       - Device release number (BCD)
    char iManufacturer;         // iManufacturer   - Index of string descriptor for the
manufacturer
    char iProduct;              // iProduct        - Index of string descriptor for the
product.
    char iSerialNumber;         // iSerialNumber   - Index of string descriptor for the
serial number.
    char bNumConfigurations;    // bNumConfigurations - Number of possible configurations
} device_dsc = {
      0x12,                   // bLength
      0x01,                   // bDescriptorType
      0x0200,                 // bcdUSB
      0x00,                   // bDeviceClass
      0x00,                   // bDeviceSubClass
      0x00,                   // bDeviceProtocol
      8,                      // bMaxPacketSize0
      USB_VENDOR_ID,          // idVendor
      USB_PRODUCT_ID,         // idProduct
      0x0001,                 // bcdDevice
      0x01,                   // iManufacturer
      0x02,                   // iProduct
      0x00,                   // iSerialNumber
      0x01                    // bNumConfigurations
  };

/* Configuration 1 Descriptor */
const char configDescriptor1[]= {
    // Configuration Descriptor
    0x09,                   // bLength             - Descriptor size in bytes
    0x02,                   // bDescriptorType     - The constant CONFIGURATION (02h)
    0x29,0x00,              // wTotalLength        - The number of bytes in the
configuration descriptor and all of its subordinate descriptors
    1,                      // bNumInterfaces      - Number of interfaces in the
configuration
    1,                      // bConfigurationValue - Identifier for Set Configuration and
Get Configuration requests
    0,                      // iConfiguration      - Index of string descriptor for the
configuration
```

```
    USB_SELF_POWER,         // bmAttributes        - Self/bus power and remote wakeup
settings
    USB_MAX_POWER,          // bMaxPower           - Bus power required in units of 2 mA

    // Interface Descriptor
    0x09,                   // bLength - Descriptor size in bytes (09h)
    0x04,                   // bDescriptorType - The constant Interface (04h)
    0,                      // bInterfaceNumber - Number identifying this interface
    0,                      // bAlternateSetting - A number that identifies a descriptor
with alternate settings for this bInterfaceNumber.
    2,                      // bNumEndpoint - Number of endpoints supported not counting
endpoint zero
    0x03,                   // bInterfaceClass - Class code
    0,                      // bInterfaceSubclass - Subclass code
    0,                      // bInterfaceProtocol - Protocol code
    0,                      // iInterface - Interface string index

    // HID Class-Specific Descriptor
    0x09,                   // bLength - Descriptor size in bytes.
    0x21,                   // bDescriptorType - This descriptor's type: 21h to indicate
the HID class.
    0x01,0x01,              // bcdHID - HID specification release number (BCD).
    0x00,                   // bCountryCode - Numeric expression identifying the country
for localized hardware (BCD) or 00h.
    1,                      // bNumDescriptors - Number of subordinate report and physical
descriptors.
    0x22,                   // bDescriptorType - The type of a class-specific descriptor
that follows
    USB_HID_RPT_SIZE,0x00,  // wDescriptorLength - Total length of the descriptor
identified above.

    // Endpoint Descriptor
    0x07,                   // bLength - Descriptor size in bytes (07h)
    0x05,                   // bDescriptorType - The constant Endpoint (05h)
    USB_HID_EP | 0x80,      // bEndpointAddress - Endpoint number and direction
    USB_TRANSFER_TYPE,      // bmAttributes - Transfer type and supplementary information
    0x40,0x00,              // wMaxPacketSize - Maximum packet size supported
    EP_IN_INTERVAL,         // bInterval - Service interval or NAK rate

    // Endpoint Descriptor
    0x07,                   // bLength - Descriptor size in bytes (07h)
    0x05,                   // bDescriptorType - The constant Endpoint (05h)
    USB_HID_EP,             // bEndpointAddress - Endpoint number and direction
    USB_TRANSFER_TYPE,      // bmAttributes - Transfer type and supplementary information
    0x40,0x00,              // wMaxPacketSize - Maximum packet size supported
    EP_OUT_INTERVAL         // bInterval - Service interval or NAK rate
};

const struct {
  char report[USB_HID_RPT_SIZE];
}hid_rpt_desc =
  {
     {0x06, 0x00, 0xFF,       // Usage Page = 0xFF00 (Vendor Defined Page 1)
      0x09, 0x01,             // Usage (Vendor Usage 1)
      0xA1, 0x01,             // Collection (Application)
  // Input report
      0x19, 0x01,             // Usage Minimum
      0x29, 0x40,             // Usage Maximum
      0x15, 0x00,             // Logical Minimum (data bytes in the report may have
minimum value = 0x00)
      0x26, 0xFF, 0x00,       // Logical Maximum (data bytes in the report may have
maximum value = 0x00FF = unsigned 255)
      0x75, 0x08,             // Report Size: 8-bit field size
      0x95, HID_INPUT_REPORT_BYTES,// Report Count
```

```
      0x81, 0x02,             // Input (Data, Array, Abs)
  // Output report
      0x19, 0x01,             // Usage Minimum
      0x29, 0x40,             // Usage Maximum
      0x75, 0x08,             // Report Size: 8-bit field size
      0x95, HID_OUTPUT_REPORT_BYTES,// Report Count
      0x91, 0x02,             // Output (Data, Array, Abs)
      0xC0}                   // End Collection
  };

//Language code string descriptor
const struct {
  char bLength;
  char bDscType;
  unsigned int string[1];
  } strd1 = {
      4,
      0x03,
      {0x0409}
    };


//Manufacturer string descriptor
const struct{
  char bLength;
  char bDscType;
  unsigned int string[6];
  }strd2={
    14,           //sizeof this descriptor string
    0x03,
    {'R','M','J','_','F','F'}
  };

//Product string descriptor
const struct{
  char bLength;
  char bDscType;
  unsigned int string[6];
}strd3={
    14,          //sizeof this descriptor string
    0x03,
    {'R','M','J','_','F','F'}
 };

//Array of configuration descriptors
const char* USB_config_dsc_ptr[1];

//Array of string descriptors
const char* USB_string_dsc_ptr[3];

void USB_Init_Desc(){
  USB_config_dsc_ptr[0] = &configDescriptor1;
  USB_string_dsc_ptr[0] = (const char*)&strd1;
  USB_string_dsc_ptr[1] = (const char*)&strd2;
  USB_string_dsc_ptr[2] = (const char*)&strd3;
}
```

Esse arquivo precisa ser incluído no projeto. No menu Project Manager ao lado direito da interface gráfica do MikroC Pro, deve-se clicar com o botão da direita em Sources, escolher “Add file to Project”, selecionar o USBdsc.c e verificar se o mesmo foi incluído ao projeto, conforme demonstra a figura 3.

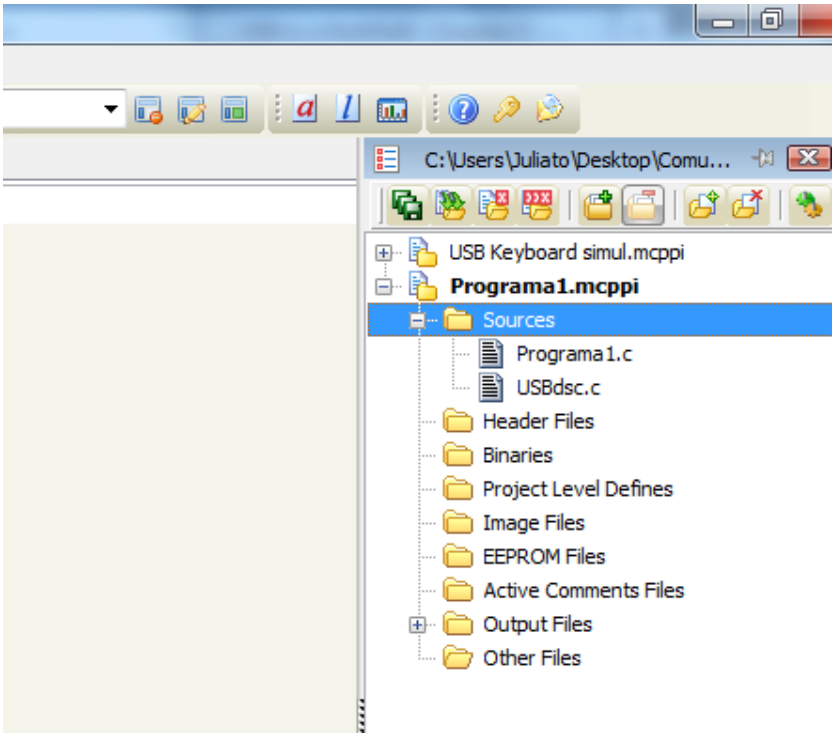

Figura 3 – Arquivos Fonte do Projeto.

Em seguida foi elaborado com base nos arquivos de exemplo da mikroElektronika o seguinte programa principal:

```
/*
================================================================
Exemplos de Programas em Linguagem C para Sistemas Embarcados

Rahul Martim Juliato (rahul.juliato@gmail.com)

Compilador: MikroC PRO v5.3

================================================================

Objetivo do programa: Estudar o envio / recebimento de
mensagens via USB.

*/



unsigned char readbuff[64] absolute 0x500; // Define tamanho do Buffer e posição
                                           // da memória
                                           // Tamanhos devem coincidir com
                                           // o USBdsc.c

unsigned char writebuff[64] absolute 0x540;

char cnt;char kk;void interrupt(){

   USB_Interrupt_Proc();       // A execução do serviço USB é executado dentro da
                               // interrupção
```

```
}

void main(void){
  ADCON1 |= 0x0F;                          // Configura o adcon1 como digital
  CMCON  |= 7;                             // Desabilita os comparadores
  HID_Enable(&readbuff,&writebuff);        // Habilita comunicação HID

  delay_ms(5000);                          // Só para dar tempo de abrir o terminal
                                           // e verificar essas mensagens
  HID_Write("Iniciou a comunicacao! ",64);
  delay_ms(10);
  HID_Write("Envie para eu repetir: ",64);

  while(1){

    while(!HID_Read());                // Lê o que for enviado via terminal

    for(cnt=0;cnt<64;cnt++)            // Carrega writebuff com readbuff
          writebuff[cnt]=readbuff[cnt];

    HID_Write("Recebi: ",64);          // Escreve...:
    while(!HID_Write(&writebuff,64));
    HID_Write("    ",64);


  }
}
```

O programa tem como função receber strings pelo terminal e enviar o que foi recebido a esse terminal. A figura 4 mostra o fluxograma do programa.

As variáveis readbuff e writebuff são utilizadas pelas funções HID_Write e HID_Read respectivamente para envio e leitura de mensagens. Essas devem possuir o mesmo comprimento declarado no arquivo descriptor, em nosso caso 64 posições, conforme a figura 1.

A execução dos serviços USB é realizada dentro de uma rotina interrupt onde deve estar a função USB_Interrupt_Proc();

No caso aqui apresentado, realizamos a configuração do microcontrolador para funcionar sem portas analógicas ou comparadores, essas configurações foram realizadas nas variáveis dos registradores ADCON1 e CMCON.

Informamos ao compilador que as variáveis de buffer de recebimento e escrita para o módulo USB são readbuffer e writebuffer respectivamente, declaradas como parâmetros da função de habilitação HID_Enable().

O delay de 5 segundos pode ser aumentado ou até mesmo retirado de acordo com a necessidade, nesse caso temos 5 segundos para configurar o terminal e receber as mensagens de boas vindas do PIC, caso a demora em conectar o terminal ao dispositivo, seja maior do que 5 segundos o funcionamento do restante do programa não será alterado, apenas não veremos as mensagens de boas vindas.

O laço infinito executa a operação lógica de acordo com o fluxograma da figura 4. O programa aguarda a leitura de uma informação, em seguida copia posição a posição os valores

lidos para um buffer de leitura e então escreve e aguarda a conclusão da rotina de escrita, voltando ao início do loop em seguida.

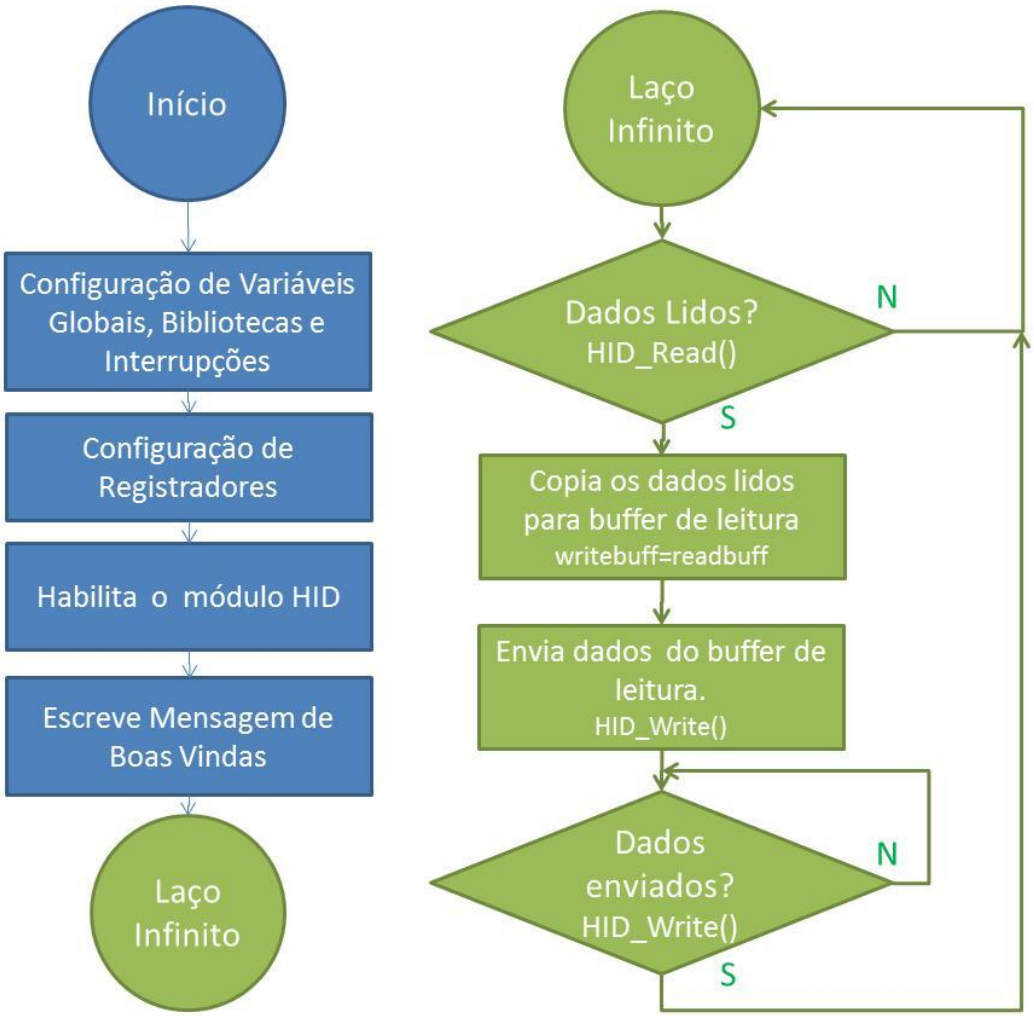

Figura 4 – Fluxograma do Programa

Para compilar o programa, não devemos nos esquecer de habilita a biblioteca USB do compilador no Library Manager, conforme a figura 5.

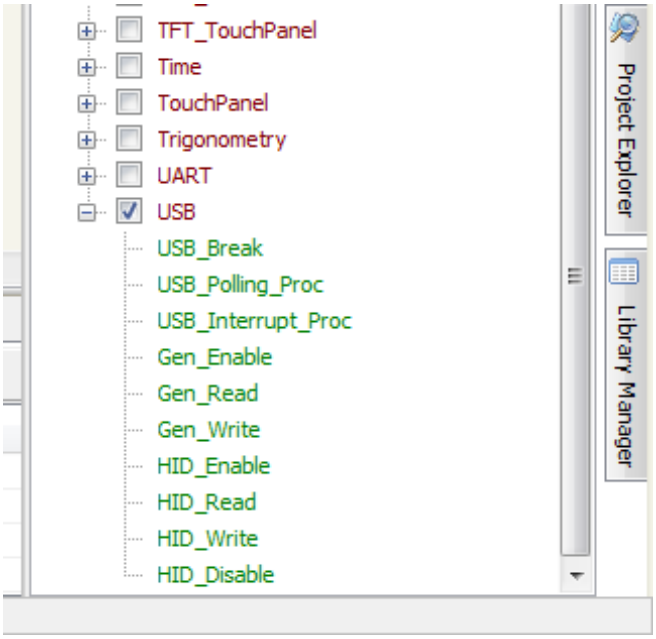

Figura 5 – Biblioteca USB Habilitada

Simulação

Para verificar o funcionamento do programa, optamos por realizar uma simulação no Proteus Isis. O circuito da figura 6 foi desenhado para executar essa função.

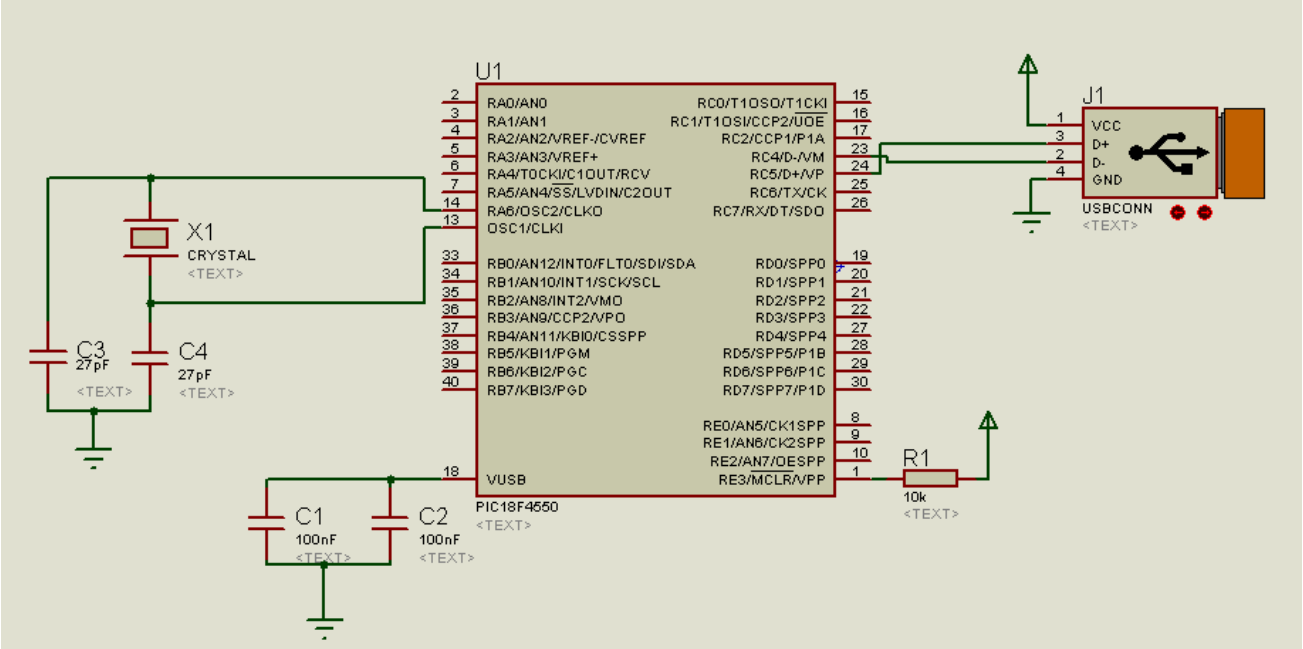

Figura 6 – Circuito de Simulação no Proteus Isis

Observação importante: tanto para a compilação como para a simulação foram utilizados clock de 4MHz.

No Isis devemos carregar ao microcontrolador o arquivo .hex gerado pelo compilador, para isso basta clicar duas vezes sobre o microcontrolador e a tela da figura 7 aparecerá, escolher o programa em “Program File” e clicar em ok.

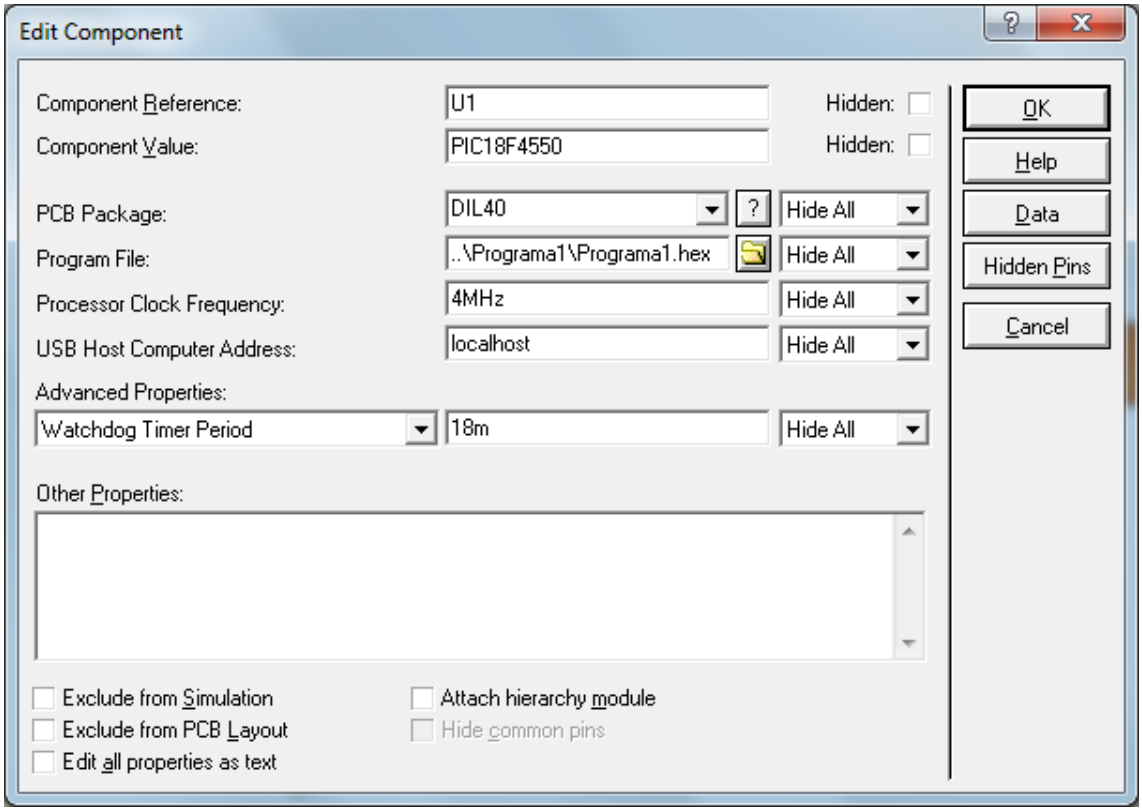

Figura 7 – Configurações do Microcontrolador no Proteus Isis

Basta então iniciar a simulação clicando no botão “play” no canto inferior esquerdo da tela.

Com a simulação em andamento, o Isis irá abrir um analisador USB chamado “Usb Analyzer”, no qual podemos verificar as comunicações de interface com o Windows conforme a figura 8.

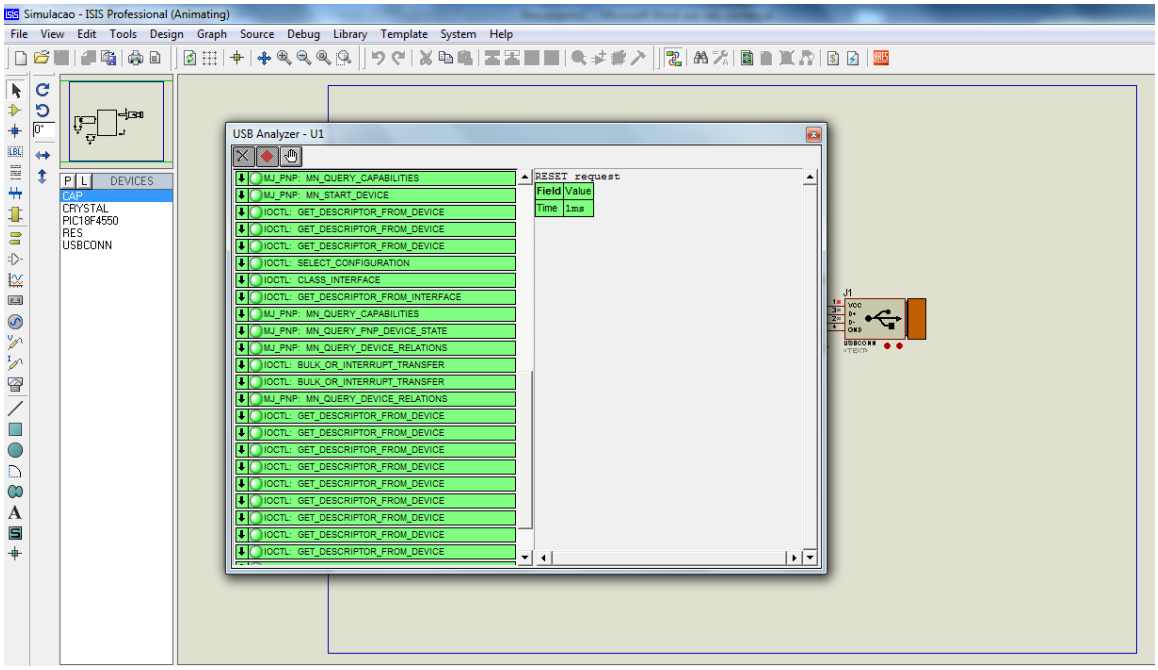

Figura 8 – Simulação em Andamento no Proteus Isis

A biblioteca de simulação do conector USB do Isis simula uma entrada real no sistema. Pode-se então verificar ao entrar na pasta de "Dispositivos e Impressoras" do Windows, conforme demonstra a figura 9 que o hardware simulado foi reconhecido com sucesso. Mais informações podem ser obtidas clicando com o botão da direita e abrindo as propriedades do dispositivo, conforme demonstram as figuras 10 e 11 respectivamente.

Convém reparar que o nome dado ao dispositivo na figura 1 aparece na figura 11.

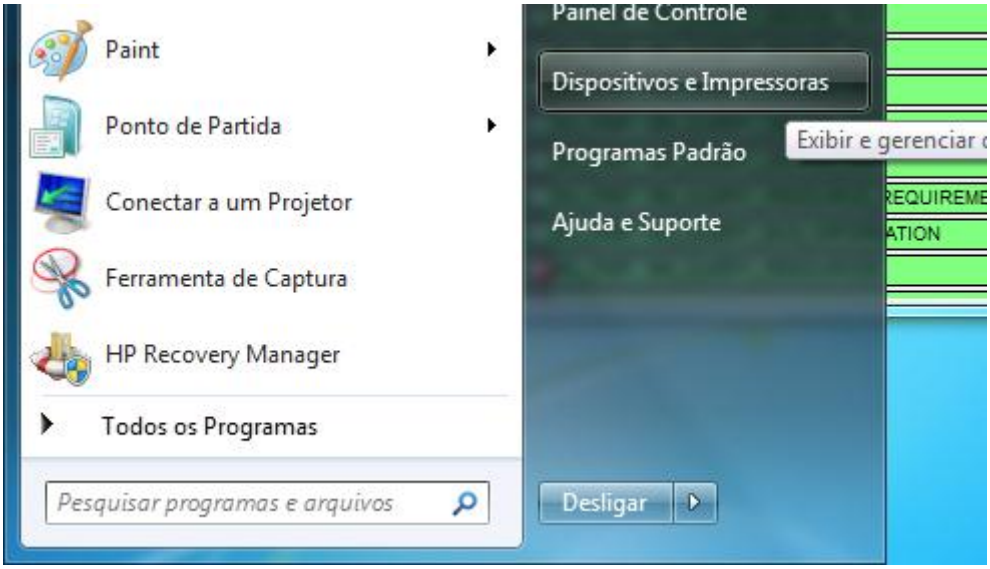

Figura 9 – Dispositivos e Impressoras no Windows 7

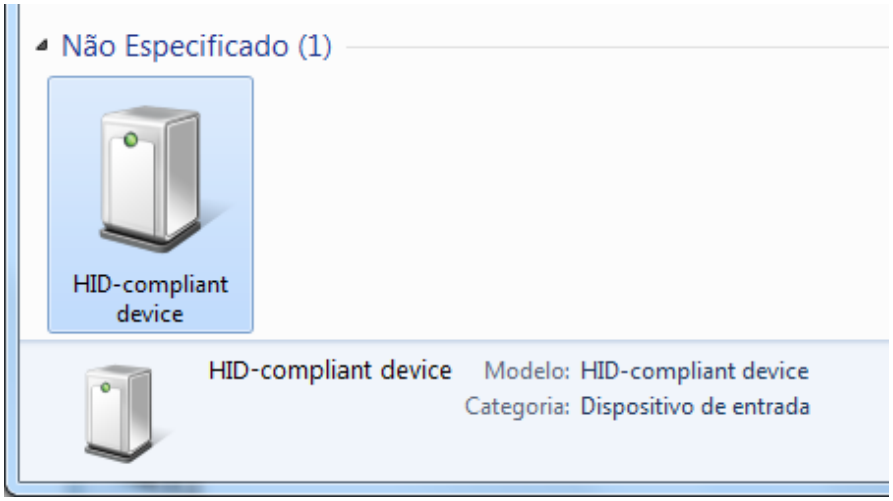

Figura 10 – Dispositivo Genérico HID detectado pelo Sistema Operacional

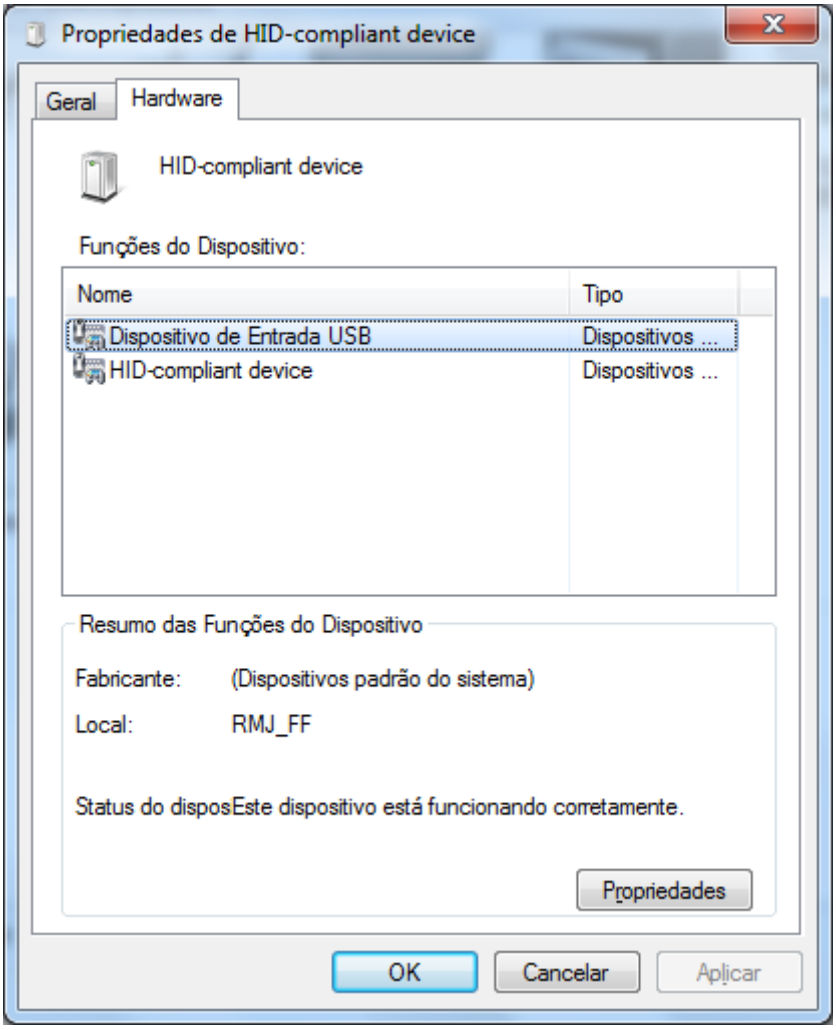

Figura 11 – Propriedades do Dispositivo Criado

Observação importante: a primeira vez em que um dispositivo HID for conectado ou simulado o Windows poderá levar um tempo para instalar os drivers.

Com a simulação em andamento, ou o circuito conectado à USB, devemos voltar ao MikroC Pro e abrir novamente o HID Terminal. Selecionar o nosso hardware na lista de "HID Devices" e verificar o programa rodando no terminal conforme mostra a figura 12.

Observação Importante: a mensagem de boas vindas aparece após 5 segundos do dispositivo conectado, caso se demore mais do que este tempo entre o início da simulação, abrir e selecionar o device no terminal a mensagem não irá aparecer, o que não significa que o circuito não está funcionando, basta aumentar o delay.

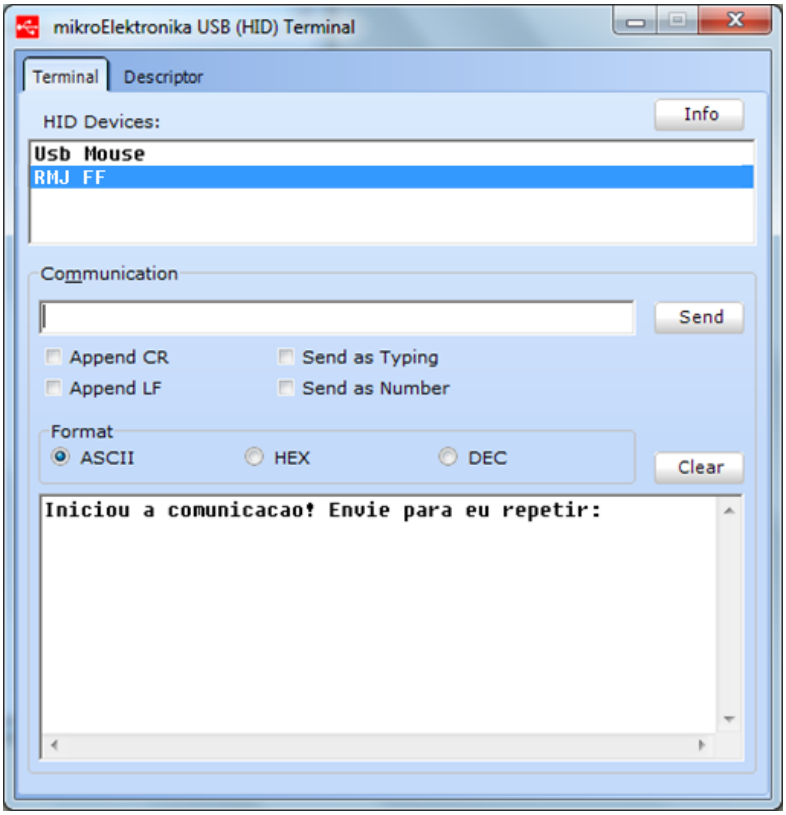

Figura 12 – HID Terminal comunicando com o Hardware USB

Basta então escrever o que se deseja enviar em “Communication” e clicar em “Send”. Se tudo ocorrer bem, a string enviada deverá aparecer na tela “terminal”. As figuras 13 e 14 mostram o envio das strings “Teste” , “Comunicação UsB” e seus devidos resultados.

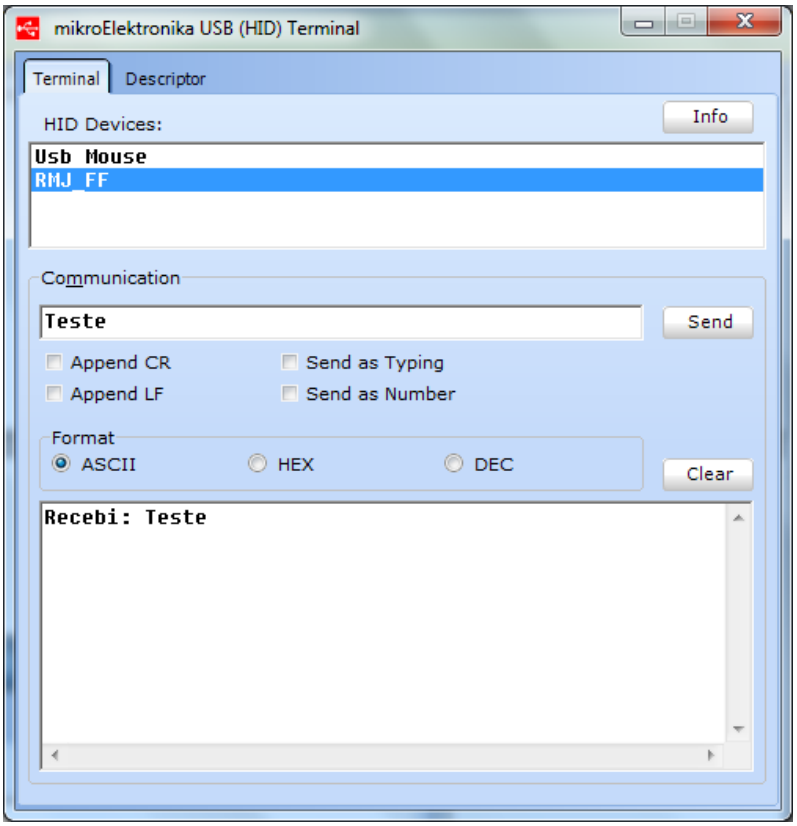

Figura 13 – String “Teste” enviada e recebida.

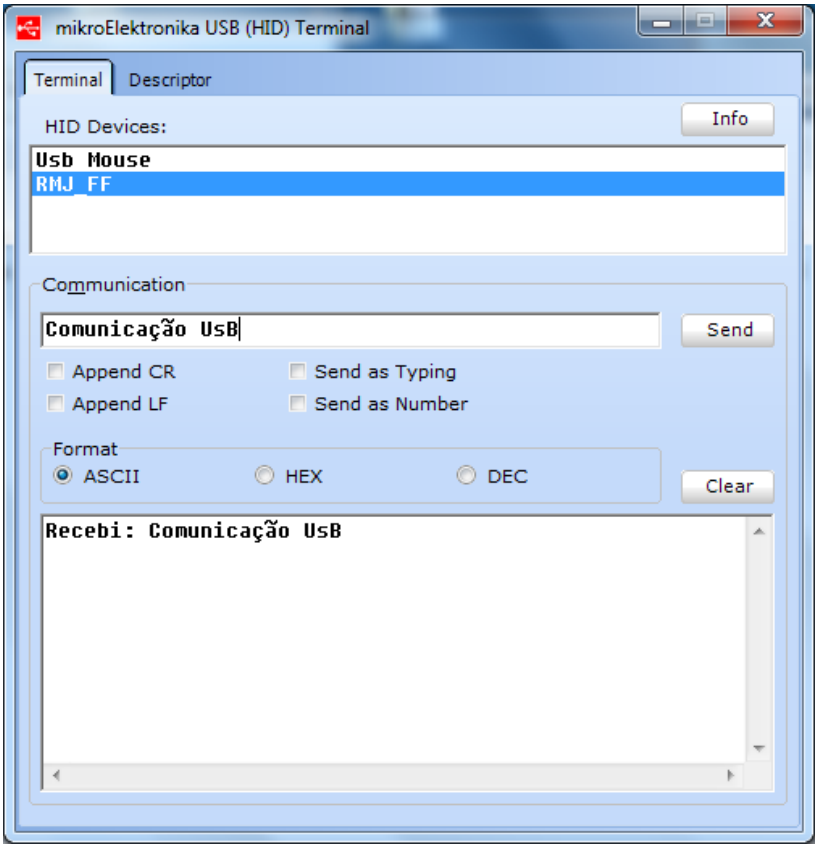

Figura 14 – String “Comunicação UsB” enviada e recebida.

Conclusões

Esses procedimentos concluem a implementação de uma comunicação simples USB de um microcontrolador PIC 18 à um PC. Vale ressaltar que ações extras devem ser tomadas caso queiramos transformar nosso hardware em um Teclado, Mouse ou Joystick, basicamente são alterações no arquivo “Descriptor” que deve ser modificado de acordo com a relação de ações descritas no http://www.usb.org/developers/hidpage/.

Caso o leitor deseje implementar a comunicação do hardware à um programa de alto nível escrito em Delphi, Visual Basic ou mesmo em Labview gerando uma interface visual elaborada, deve-se utilizar bibliotecas e dlls especificas para trabalhar com terminais HID.

Referências e Agradecimentos

Esse artigo talvez não fosse escrito sem a insistência de alguns amigos que me viram sofrer um pouco, errar bastante, até encontrar um caminho funcional. Meus agradecimentos aos engenheiros Rodolfo Carlos Blumel e Francisco Fambrini.

Os seguintes artigos me foram muito úteis, não deixe de lê-los! Meus agradecimentos aos seus autores:

- PIC Communication with USB By Roy Seifert ;
- USB in a NutShell - **http://www.beyondlogic.org/usbnutshell/usb1.shtml**;
- Microchip Application Note 1140;
- Comunicação USB com o PIC - Vitor Amadeu Souza.