USB Controlled Stepper Motor Interface

MUDIT AGARWAL

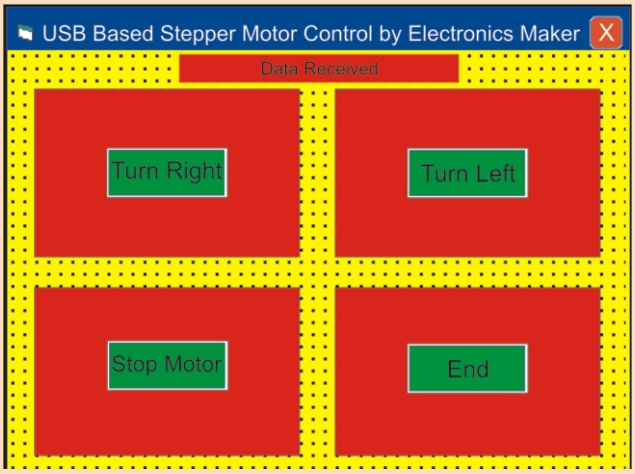
MUDIT AGARWAL

The Universal Serial Bus is one of the most common interfaces used in electronic consumer products today, including PCs, cameras, GPS devices, MP3 players, modems, printers, and scanners etc. The USB was originally developed by Compaq, Microsoft, Intel, and NEC, and later by Hewlett-Packard, Lucent, and Philips as well. These companies eventually formed the nonprofit corporation USB Implementers Forum Inc. to organize the development and publication of USB specifications. In this article we shows how to control a stepper motor with a USB port of a PC with the help of PIC microcontroller.

The USB is a high-speed serial interface that can also provide power to devices connected to it. A USB bus supports up to 127 devices connected through a four-wire serial cable of up to three or even five meters in length. Many USB devices can be connected to the same bus with hubs, which can have 4, 8, or even 16 ports. A device can be plugged into a hub which is plugged into another hub, and so on.

USB Signal

USB signals are bi-phase, and signals are sent from the host computer using the NRZI data encoding technique. In this technique, the signal levels inverted for each change to a logic 0. The signal level for a logic 1 is not changed. A 0 bit is "stuffed" after every six consecutive ones in the data stream to make the data dynamic (this is called bit stuffing because the extra bit lengthens the data stream). A packet of data transmitted by the host is sent to every device connected to the bus, traveling downward through the chain of hubs. All the devices receive the signal, but only one of them, the addressed one, accepts the data. Conversely, only one device at any time can transmit to the host, and the data travels upward through the chain of hubs until it reaches the host. USB devices attached to the bus may be full-custom devices, requiring a full-custom device driver, or they may belong to a device class.



Deviced classes enable the same device driver to be used for several devices having similar functionalities. For example, a printer device has the device class 0x07, and most printers use drivers of this type.

Endpoint: An endpoint is either a source or a sink of data. A single USB device can have a number of endpoints, the limit being sixteen IN and sixteen OUT endpoints.

Transaction: A transaction is a transfer of data on the bus.

Pipe: A pipe is a logical data connection between the host and an endpoint.

USB States

Idle: The bus is in idle state when the pulled-up line is high and the other line is low. This is the state of the lines before and after a packet transmission.

Detached: When no device is connected to the bus, the host sees both lines as low.

Attached: When a device is connected to the bus, the host sees either D+ or D_ go to logic high, which means a device has been plugged in.

J state: The same as idle state.

K state: The opposite of J state.

SE0: The single ended zero state, where both lines on the bus are pulled low.

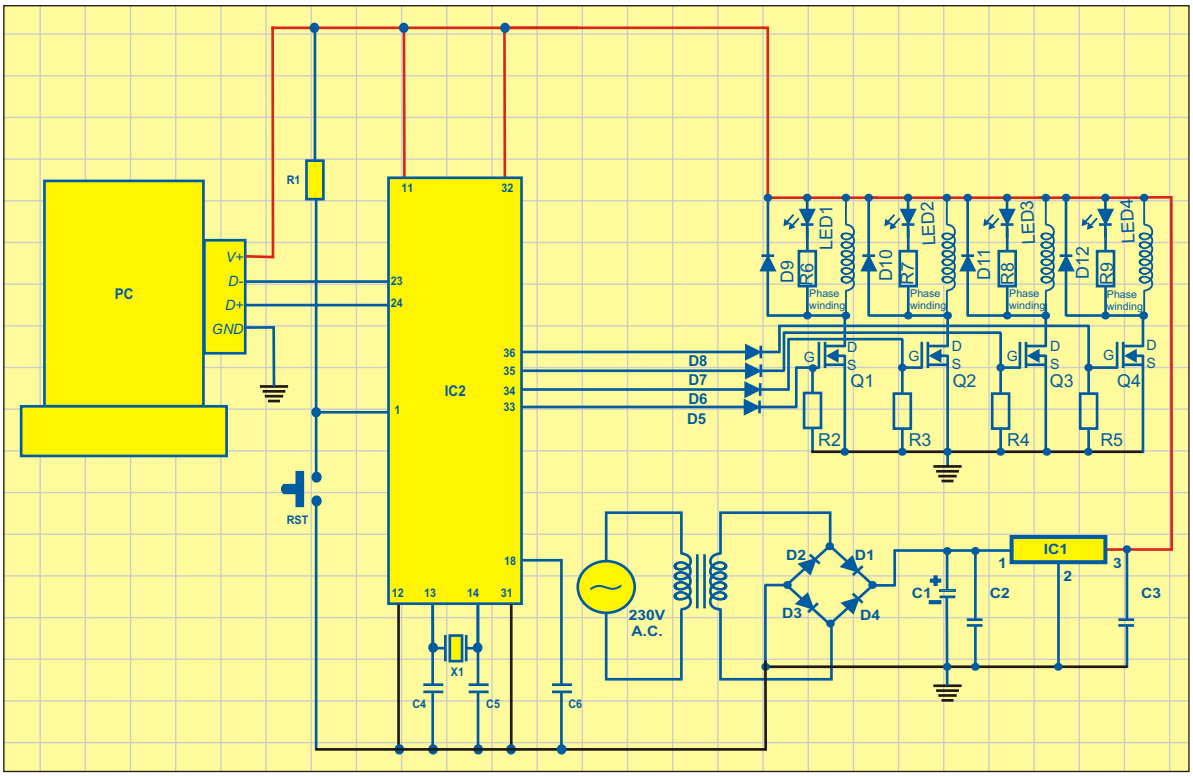


Fig. 1.Circuit Diagram of USB controlled Stepper Motor Interface

SE1: The single ended one state, where both lines on the bus are high. SE1 is an illegal condition on the bus; it must never be in this state.

Reset: When the host wants to communicate with a device on the bus, it first sends a "reset" condition by pulling low both data lines (SE0 state) for at least10ms.

EOP: The end of packet state, which is basically an SE0 state for 2 bit times, followed by a J state for 1 bit time.

Keep alive: The state achieved by EOP. Keep alive is sent at least once every millisecond to keep the device from suspending.

Suspend: Used to save power, suspend is implemented by not sending anything to a device for 3ms. A suspended device draws less than 0.5mA from the bus and must recognize reset and

Sync	PID	ADDR	ENDP	CRC	EOP
	8 bits	7 bits	4 bits	5 bits	

resume signals.

Resume: A suspended device is woken up by reversing the polarity of the signal on the data lines for at least 20ms, followed by a low-speed EOP signal.

Data Transmission

Data is transmitted on a USB bus in packets. A packet starts with a sync pattern to allow the receiver clock to synchronize with the data. A packet identifier (PID) byte immediately follows the sync field of every USB packet. A PID itself is 4 bits long, and the 4 bits are repeated in a complemented form. There are seventeen different PID values There are four packet formats, based on

PID Type	PID name	Bits	Description
Token	OUT	11100001	Host to device transaction
	IN	01101001	Device to host transaction
	SOF	10100101	Start of frame
	SETUP	00101101	Setup command
Data	DATA0	11000011	Data packet PID even
	DATA1	01001011	Data packet PID odd
	DATA2	10000111	Data packet PID high speed
	MDATA	00001111	Data packet PID high speed
Handshake	ACK	11010010	Receiver accepts packet
	NAK	01011010	Receiver does not accept packet
	STALL	00011110	Stalled
	NYET	00111100	No response from receiver
Special	PRE	00111100	Host preample
	ERR	01111000	Split transaction error
	SPLIT	01111000	High speed split transaction
	PING	10110100	High speed flow control
	Reserved	11110000	Reserved

which PID is at the start of the packet: token packets, data packets, handshake packets, and special packets.

Enumeration

When a device is plugged into a USB bus, it becomes known to the host through a process

called enumeration. The steps of enumeration are, When a device is plugged in, the host becomes aware of it. The host sends a USB reset signal to the device to place the device in a known state. The reset device responds to address 0. The host sends a request on address 0 to the device to find out its maximum packet size using a Get Descriptor command. The device responds by sending a small portion of the device descriptor. The host sends a USB reset again. The host assigns a unique address to the device and sends a Set Address request to the device. After the request is completed, the device assumes the new address. At this point the host is free to reset any other newly plugged-in devices on the bus. The host sends a Get Device Descriptor request to retrieve the complete device descriptor, gathering information such as manufacturer, type of device, and maximum control packet size. The host sends a Get Configuration Descriptors request to receive the device's configuration data, such as power requirements and the types and number of interfaces supported. The host may request any additional descriptors from the device.

The most common USB descriptors are:

Device descriptors, Configuration descriptors Interface descriptors, HID descriptors, Endpoint descriptors. Here i cannot discuss all these things because of space limitation.

PIC18F4550 microcontroller is use for USB communication with PC. PORTC pins RC4 (pin 23) and RC5 (pin 24) are used for USB interface. RC4 is the USB data D_ pin, and RC5 is the USB data D+ pin. Internal pull-up resistors are provided which can be disabled (setting UPUEN =0) if desired and external pull-up resistors can be used instead. For full-speed operation an internal or external resistor should be connected to data pin D+, and for low-speed operation an internal or external resistor should be connected to data pin D-. Operation of the USB module is configured using three control registers, and a total of twenty-two registers are used to manage the actual USB transactions The operation of the USB link requires the microcontroller to keep the connection alive by sending keep-alive messages to the PC every several milliseconds. This is achieved by setting up a timer interrupt service routine using

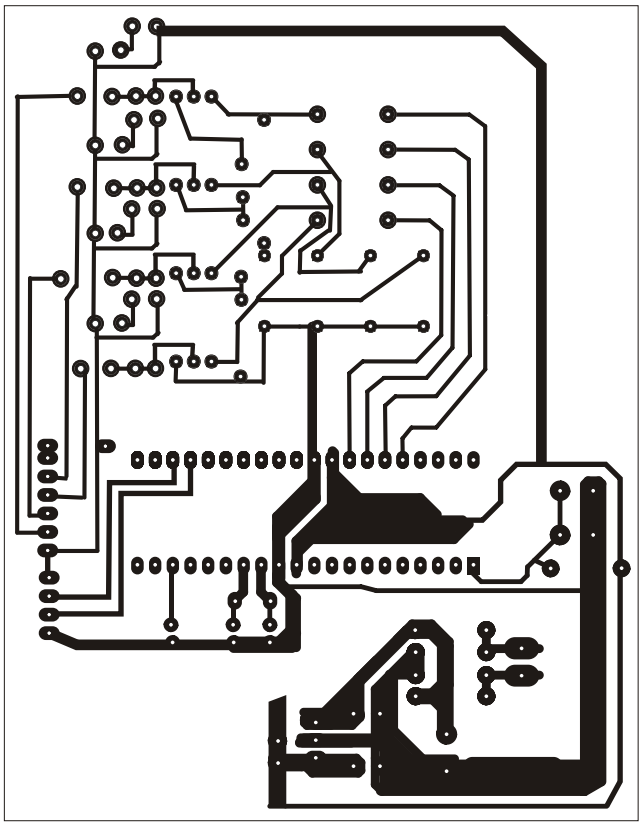


Fig. 2. PCB Layout of USB controlled Stepper Motor Interface. (92% of actual size)

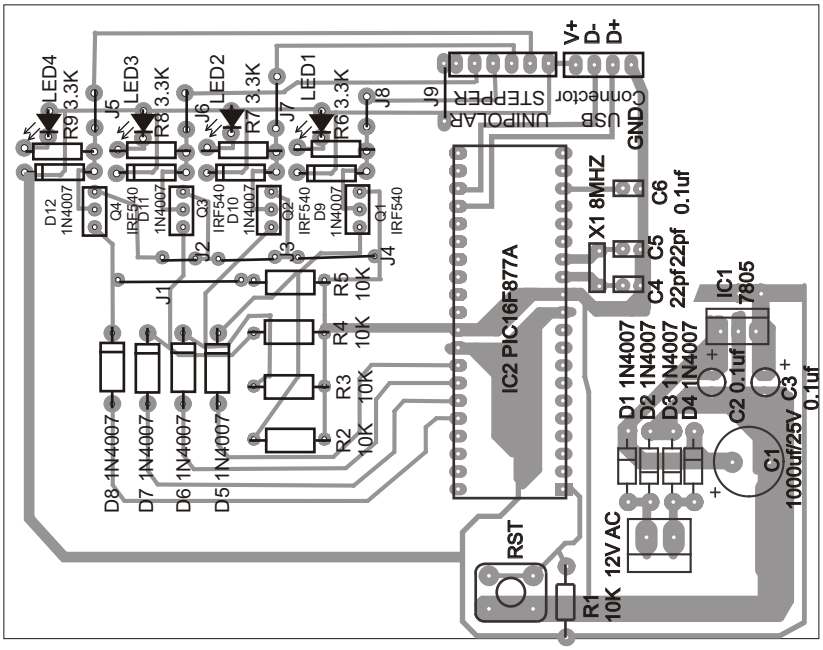


Fig. 3.Component Layout of USB controlled Stepper Motor Interface. (92% of actual size)

CONSTRUCTION

TIMER 0. Timer TMR0L is reloaded and timer interrupts are re-enabled just before returning from the interrupt service routine. Inside the main program PORTB is defined as digital I/O and TRISB is cleared to 0 so all PORTB pins are outputs. All the interrupt registers are then set to their power-on-reset values for safety. The timer interrupts are then set up. The timer is operated in 8-bit mode with a prescaler of 256. Although the crystal clock frequency is 8MHZ, the CPU is operated with a 48MHz clock, *by enabling its PLL*. Selecting a timer value of TMR0L = 100 with a 48MHz clock gives timer interrupt intervals of 3.3ms.

Circuit Diagram of USB controlled Stepper Motor Interface is show in fig. 1. PCB layout of USB controlled Stepper Motor Interface is show in fig. 2. Component layout of USB controlled Stepper Motor Interface is show in fig. 3.

The Software Running On Microcontroller

```
Unsigned char Read_buffer[64];
unsigned char Write_buffer[64];
unsigned char num;
unsigned char const HID_INPUT_REPORT_BYTES
= 1;
unsigned char const HID_OUTPUT_REPORT_BYTES
= 1;
unsigned char const HID_FEATURE_REPORT_BYTES
= 2;
unsigned char const NUM_ENDPOINTS =
2;
unsigned char const ConfigDescr_wTotalLength =
USB_CONFIG_DESCRIPTOR_LEN +
USB_INTERF_DESCRIPTOR_LEN +
USB_HID_DESCRIPTOR_LEN +
(NUM_ENDPOINTS *
USB_ENDP_DESCRIPTOR_LEN);
unsigned char const HID_ReportDesc_len =
47;
unsigned char const Low_HID_ReportDesc_len =
HID_ReportDesc_len;
unsigned char const High_HID_ReportDesc_len =
HID_ReportDesc_len >> 8;
unsigned char const Low_HID_PACKET_SIZE =
HID_PACKET_SIZE;
unsigned char const High_HID_PACKET_SIZE
=
HID_PACKET_SIZE >> 8;
unsigned char const
  DeviceDescr[USB_DEVICE_DESCRIPTOR_LEN*2]
= {
USB_DEVICE_DESCRIPTOR_LEN, 0,
USB_DEVICE_DESCRIPTOR_TYPE, 0,
0x00, 0,0x02, 0,0x00, 0,0x00, 0, 0x00, 0,
EP0_PACKET_SIZE, 0,0x34, 0,0x12, 0,0x01, 0,
0x00, 0,
0x01, 0,0x00, 0, 0x01, 0, 0x02, 0,0x00, 0,0x01, 0
};
unsigned char const
 ConfigDescr[USB_CONFIG_DESCRIPTOR_LEN*2]
= {
USB_CONFIG_DESCRIPTOR_LEN, 0,
USB_CONFIG_DESCRIPTOR_TYPE, 0,
ConfigDescr_wTotalLength, 0,
0x00, 0, 0x01, 0, 0x01, 0,0x00, 0,0xA0, 0,50, 0
};
unsigned char const
  InterfaceDescr[USB_INTERF_DESCRIPTOR_LEN*2]
= {
USB_INTERF_DESCRIPTOR_LEN, 0,
USB_INTERFACE_DESCRIPTOR_TYPE, 0,
0x00, 0, 0x00, 0, NUM_ENDPOINTS, 0,0x03, 0,
0x00, 0, 0x00, 0, 0x00, 0};
unsigned char const
 HID_Descriptor[USB_HID_DESCRIPTOR_LEN*2] =
{
USB_HID_DESCRIPTOR_LEN, 0,
USB_HID_DESCRIPTOR_TYPE, 0,
0x01, 0, 0x01, 0,0x00, 0,0x01, 0, 0x22, 0,
Low_HID_ReportDesc_len,
0,High_HID_ReportDesc_len, 0};
unsigned char const
 EP1_RXDescr[USB_ENDP_DESCRIPTOR_LEN*2] =
{
USB_ENDP_DESCRIPTOR_LEN, 0,
USB_ENDPOINT_DESCRIPTOR_TYPE, 0,
0x81, 0,USB_ENDPOINT_TYPE_INTERRUPT, 0,
Low_HID_PACKET_SIZE,
0,High_HID_PACKET_SIZE, 0, 0 };
unsigned char const
 EP1_TXDescr[USB_ENDP_DESCRIPTOR_LEN*2] =
{
USB_ENDP_DESCRIPTOR_LEN, 0,
USB_ENDPOINT_DESCRIPTOR_TYPE, 0,
0x01, 0,USB_ENDPOINT_TYPE_INTERRUPT, 0,
Low_HID_PACKET_SIZE, 0,HID_PACKET_SIZE, 0,1,
0};
unsigned char const
 HID_ReportDesc[HID_ReportDesc_len*2] = {0x06,
0,0xA0, 0,
0xFF, 0,0x09, 0,0x01, 0,0xA1, 0,0x01, 0,0x09,
0,0x03, 0,0x15, 0,0x00, 0,0x26, 0,0x00, 0,0xFF,
0,0x75, 0,0x08, 0,0x95, 0,
HID_INPUT_REPORT_BYTES, 0,0x81, 0, 0x02,
0,0x09, 0,
0x04, 0,0x15, 0,0x00, 0,0x26, 0, 0x00, 0,0xFF,
0,0x75, 0,
0x08, 0,0x95, 0,HID_OUTPUT_REPORT_BYTES, 0,
0x91, 0, 0x02, 0,0x09, 0, 0x05, 0,0x15, 0,0x00,
0,0x26, 0,0x00, 0,0xFF, 0,0x75, 0,0x08, 0,0x95,
0,HID_FEATURE_REPORT_BYTES,0,0xB1, 0, 0x02,
0,0xC0, 0};
unsigned char const LangIDDescr[8] = {0x04, 0,
USB_STRING_DESCRIPTOR_TYPE, 0,0x09, 0,0x04,
0};
unsigned char const ManufacturerDescr[24] = {12,
0,
USB_STRING_DESCRIPTOR_TYPE, 0,  'M', 0, 0, 0,
'u', 0, 0, 0,  'd', 0, 0, 0,  'i', 0, 0, 0,  't', 0, 0, 0};
unsigned char const ProductDescr[32] = { 16, 0,
 USB_STRING_DESCRIPTOR_TYPE, 0, 'A', 0, 0, 0,
 'g', 0, 0, 0, 'a', 0, 0, 0, 'r', 0, 0, 0, 'w', 0, 0, 0, 'a', 0, 0,
0, 'l', 0, 0, 0};
unsigned char const StrUnknownDescr[4] = {
 2, 0, USB_STRING_DESCRIPTOR_TYPE, 0};
void interrupt()
{HID_InterruptProc(); TMR0L = 100;
INTCON.TMR0IF = 0; }
void InitUSBdsc()
{Byte_tmp_0[0] = NUM_ENDPOINTS;
Byte_tmp_0[0] = ConfigDescr_wTotalLength;
Byte_tmp_0[0] = HID_ReportDesc_len;
Byte_tmp_0[0] = Low_HID_ReportDesc_len;
Byte_tmp_0[0] = High_HID_ReportDesc_len;
Byte_tmp_0[0] = Low_HID_PACKET_SIZE;
Byte_tmp_0[0] = High_HID_PACKET_SIZE;
DeviceDescr;ConfigDescr;InterfaceDescr;HID_Desc
riptor;
EP1_RXDescr;EP1_TXDescr;HID_ReportDesc;
LangIDDescr;ManufacturerDescr;ProductDescr;
StrUnknownDescr;}
void main()
{ADCON1 = 0xFF;TRISB = 0; PORTB = 0;
INTCON=0;
INTCON2=0xF5;RCON.IPEN=0;PIE1=0;PIE2=0;
PIR1=0;
PIR2=0;T0CON = 0x47; TMR0L = 100;
INTCON.TMR0IE = 1;
T0CON.TMR0ON = 1; INTCON = 0xE0;
Hid_Enable(&Read_buffer,
&Write_buffer);Delay_ms(1000);
Delay_ms(1000);for(;;) {num=0;while(num != 4)
{num = Hid_Read();}
if(Read_buffer[0] == 'P' && Read_buffer[1] == '='
&& Read_buffer[3] == 'T'){PORTB =
Read_buffer[2];}}
Hid_Disable();}
```

The Software on PC

```
Option Explicit
Declare Function hidConnect Lib "mcHID.dll" Alias
"Connect" (ByVal pHostWin As Long) As Boolean
Declare Function hidDisconnect Lib "mcHID.dll" Alias
"Disconnect" () As Boolean
Declare Function hidGetItem Lib "mcHID.dll" Alias
"GetItem" (ByVal pIndex As Long) As Long
Declare Function hidGetItemCount Lib "mcHID.dll"
Alias "GetItemCount" () As Long
Declare Function hidRead Lib "mcHID.dll" Alias
"Read" (ByVal pHandle As Long, ByRef pData As Byte)
As Boolean
Declare Function hidWrite Lib "mcHID.dll" Alias
"Write" (ByVal pHandle As Long, ByRef pData As Byte)
As Boolean
Declare Function hidReadEx Lib "mcHID.dll" Alias
"ReadEx" (ByVal pVendorID As Long, ByVal pProductID
As Long, ByRef pData As Byte) As Boolean
Declare Function hidWriteEx Lib "mcHID.dll" Alias
"WriteEx" (ByVal pVendorID As Long, ByVal
pProductID As Long, ByRef pData As Byte) As Boolean
Declare Function hidGetHandle Lib "mcHID.dll" Alias
"GetHandle" (ByVal pVendoID As Long, ByVal
pProductID As Long) As Long
```

```
Declare Function hidGetVendorID Lib "mcHID.dll" Alias "GetVendorID" (ByVal pHandle As Long) As Long
Declare Function hidGetProductID Lib "mcHID.dll" Alias "GetProductID" (ByVal pHandle As Long) As Long
Declare Function hidGetVersion Lib "mcHID.dll" Alias "GetVersion" (ByVal pHandle As Long) As Long
Declare Function hidGetVendorName Lib "mcHID.dll" Alias "GetVendorName" (ByVal pHandle As Long, ByVal pText As String, ByVal pLen As Long) As Long
Declare Function hidGetProductName Lib "mcHID.dll" Alias "GetProductName" (ByVal pHandle As Long, ByVal pText As String, ByVal pLen As Long) As Long
Declare Function hidGetSerialNumber Lib "mcHID.dll" Alias "GetSerialNumber" (ByVal pHandle As Long, ByVal pText As String, ByVal pLen As Long) As Long
Declare Function hidGetInputReportLength Lib "mcHID.dll" Alias "GetInputReportLength" (ByVal pHandle As Long) As Long
Declare Function hidGetOutputReportLength Lib "mcHID.dll" Alias "GetOutputReportLength" (ByVal pHandle As Long) As Long
Declare Sub hidSetReadNotify Lib "mcHID.dll" Alias "SetReadNotify" (ByVal pHandle As Long, ByVal pValue As Boolean)
Declare Function hidIsReadNotifyEnabled Lib "mcHID.dll" Alias "IsReadNotifyEnabled" (ByVal pHandle As Long) As Boolean
Declare Function hidIsAvailable Lib "mcHID.dll" Alias "IsAvailable" (ByVal pVendorID As Long, ByVal pProductID As Long) As Boolean
Private Declare Function CallWindowProc Lib "user32" Alias "CallWindowProcA" (ByVal lpPrevWndFunc As Long, ByVal hwnd As Long, ByVal Msg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
Private Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA" (ByVal hwnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
Private Const WM_APP = 32768
Private Const GWL_WNDPROC = -4
Private Const WM_HID_EVENT = WM_APP + 200
Private Const NOTIFY_PLUGGED = 1
Private Const NOTIFY_UNPLUGGED = 2
Private Const NOTIFY_CHANGED = 3
Private Const NOTIFY_READ = 4
Private FPrevWinProc As Long
Private FWinHandle As Long
Public Function ConnectToHID(ByVal pHostWin As Long) As Boolean
FWinHandle = pHostWin
ConnectToHID = hidConnect(FWinHandle)
FPrevWinProc = SetWindowLong(FWinHandle, GWL_WNDPROC, AddressOf WinProc)
End Function
Public Function DisconnectFromHID() As Boolean
DisconnectFromHID = hidDisconnect
SetWindowLong FWinHandle, GWL_WNDPROC, FPrevWinProc
End Function
Private Function WinProc(ByVal pHWnd As Long, ByVal pMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
If pMsg = WM_HID_EVENT Then
Select Case wParam
Case Is = NOTIFY_PLUGGED
MainForm.OnPlugged (lParam)
Case Is = NOTIFY_UNPLUGGED
MainForm.OnUnplugged (lParam)
Case Is = NOTIFY_CHANGED
MainForm.OnChanged
Case Is = NOTIFY_READ
MainForm.OnRead (lParam)
End Select End If
WinProc = CallWindowProc(FPrevWinProc, pHWnd, pMsg, wParam, lParam)
End Function
Dim i, ii, j As Variant
Private Const VendorID = 4660
Private Const ProductID = 1
Private Const BufferInSize = 4
Private Const BufferOutSize = 4
Dim BufferIn(0 To BufferInSize) As Byte
Dim BufferOut(0 To BufferOutSize) As Byte
Private Sub Command1_Click()
For i = 0 To 9999
BufferOut(0) = 0 ' first by is always the report ID
BufferOut(1) = Asc("P") ' first data item ("P")
BufferOut(2) = Asc("=") ' second data item ("-")
BufferOut(3) = Asc("a")
BufferOut(4) = Asc("T") ' fourth data item ("T")
hidWriteEx VendorID, ProductID, BufferOut(0)
lblstatus = "Turn Right"
For ii = 0 To 100
DoEvents
Next
BufferOut(0) = 0
BufferOut(1) = Asc("P") ' first data item ("P")
BufferOut(2) = Asc("=") ' second data item ("-")
BufferOut(3) = Asc("b")
BufferOut(4) = Asc("T") ' fourth data item ("T")
hidWriteEx VendorID, ProductID, BufferOut(0)
hidWriteEx VendorID, ProductID, BufferOut(1)
hidWriteEx VendorID, ProductID, BufferOut(2)
hidWriteEx VendorID, ProductID, BufferOut(3)
hidWriteEx VendorID, ProductID, BufferOut(4)
For ii = 0 To 100
DoEvents
Next
BufferOut(0) = 0 ' first by is always the report ID
BufferOut(1) = Asc("P") ' first data item ("P")
BufferOut(2) = Asc("=") ' second data item ("-")
BufferOut(3) = Asc("d")
BufferOut(4) = Asc("T") ' fourth data item ("T")
hidWriteEx VendorID, ProductID, BufferOut(0)
hidWriteEx VendorID, ProductID, BufferOut(1)
hidWriteEx VendorID, ProductID, BufferOut(2)
hidWriteEx VendorID, ProductID, BufferOut(3)
hidWriteEx VendorID, ProductID, BufferOut(4)
For ii = 0 To 100
DoEvents
Next
BufferOut(0) = 0 ' first by is always the report ID
BufferOut(1) = Asc("P") ' first data item ("P")
BufferOut(2) = Asc("=") ' second data item ("-")
BufferOut(3) = Asc("h")
BufferOut(4) = Asc("T") ' fourth data item ("T")
hidWriteEx VendorID, ProductID, BufferOut(0)
hidWriteEx VendorID, ProductID, BufferOut(1)
hidWriteEx VendorID, ProductID, BufferOut(2)
hidWriteEx VendorID, ProductID, BufferOut(3)
hidWriteEx VendorID, ProductID, BufferOut(4)
For ii = 0 To 100
DoEvents
Next
Next
End Sub
Private Sub Command2_Click()
For i = 0 To 9999
BufferOut(0) = 0 ' first by is always the report ID
BufferOut(1) = Asc("P") ' first data item ("P")
BufferOut(2) = Asc("=") ' second data item ("-")
BufferOut(3) = Asc("h")
BufferOut(4) = Asc("T") ' fourth data item ("T")
HidWriteEx VendorID, ProductID, BufferOut(0)
hidWriteEx VendorID, ProductID, BufferOut(1)
hidWriteEx VendorID, ProductID, BufferOut(2)
hidWriteEx VendorID, ProductID, BufferOut(3)
hidWriteEx VendorID, ProductID, BufferOut(4)
lblstatus = "Turn Left"
For ii = 0 To 100
DoEvents
Next
BufferOut(0) = 0 ' first by is always the report ID
BufferOut(1) = Asc("P") ' first data item ("P")
BufferOut(2) = Asc("=") ' second data item ("-")
BufferOut(3) = Asc("d") ' 0110 0010
BufferOut(4) = Asc("T") ' fourth data item ("T")
hidWriteEx VendorID, ProductID, BufferOut(0)
hidWriteEx VendorID, ProductID, BufferOut(1)
hidWriteEx VendorID, ProductID, BufferOut(2)
hidWriteEx VendorID, ProductID, BufferOut(3)
hidWriteEx VendorID, ProductID, BufferOut(4)
For ii = 0 To 100
DoEvents
Next
BufferOut(0) = 0 ' first by is always the report ID
BufferOut(1) = Asc("P") ' first data item ("P")
BufferOut(2) = Asc("=") ' second data item ("-")
BufferOut(3) = Asc("a") ' 0110 1000
BufferOut(4) = Asc("T") ' fourth data item ("T")
hidWriteEx VendorID, ProductID, BufferOut(0)
hidWriteEx VendorID, ProductID, BufferOut(1)
hidWriteEx VendorID, ProductID, BufferOut(2)
hidWriteEx VendorID, ProductID, BufferOut(3)
hidWriteEx VendorID, ProductID, BufferOut(4)
For ii = 0 To 100
DoEvents
Next
Next
End Sub
Private Sub Command3_Click()
BufferOut(0) = 0 ' first byte is always the report ID
BufferOut(1) = Asc("P") ' first data item ("P")
BufferOut(2) = Asc("=") ' second data item ("=")
BufferOut(3) = Asc("@") ' third data item ("@")
BufferOut(4) = Asc("T") ' fourth data item ("T")
' write the data (don't forget, pass the whole array)...
hidWriteEx VendorID, ProductID, BufferOut(0)
hidWriteEx VendorID, ProductID, BufferOut(1)
hidWriteEx VendorID, ProductID, BufferOut(2)
hidWriteEx VendorID, ProductID, BufferOut(3)
hidWriteEx VendorID, ProductID, BufferOut(4)
'lblstatus = "Stop Motor"
End Sub
Private Sub Command4_Click()
Form_Unload (0)
End
End Sub
Private Sub Form_Load()
 ConnectToHID (Me.hwnd)
End Sub
Private Sub Form_Unload(Cancel As Integer)
 DisconnectFromHID
End Sub
Public Sub OnPlugged(ByVal pHandle As Long)
If hidGetVendorID(pHandle) = VendorID And hidGetProductID(pHandle) = ProductID Then
End If
End Sub
If hidGetVendorID(pHandle) = VendorID And hidGetProductID(pHandle) = ProductID Then
 End If
End Sub
Public Sub OnChanged()
Dim DeviceHandle As Long
DeviceHandle = hidGetHandle(VendorID, ProductID)
 hidSetReadNotify DeviceHandle, True
End Sub
Public Sub OnRead(ByVal pHandle As Long)
 If hidRead(pHandle, BufferIn(0)) Then
If (BufferIn(1) = Asc("P") And BufferIn(2) = Asc("=") And BufferIn(4) = Asc("T")) Then
txtreceived = BufferIn(3)
Label1.Caption = Str$(BufferIn(3))
End If
End If
End Sub
Public Sub WriteSomeData()
 BufferOut(0) = 0  ' first by is always the report ID
 BufferOut(1) = 10  ' first data item, etc etc
HidWriteEx VendorID, ProductID, BufferOut(0)
End Sub
```