Kommunikations-DLL

Mit der Kommunikations-DLL können von jedem beliebigen Programm aus die Ereignisse an der Tastatur-Matrix und den Eingabe-Ports des USB-Controllers abgefragt und Daten zu den Ausgabeports gesendet werden.

Die Kommunikation erfolgt nach der Initialisierung des USB-Gerätes über zwei Datenpuffer. Der Schreibpuffer wird vor der Transferfunktion mit Daten für die Ausgabe-Ports gefüllt. Im Lesepuffer werden u.a. die Daten der Tastaturmatrix und der Eingabe-Ports zurückgeliefert.

Der Controller Manager sowie die Tools zur Programmierung kommunizieren mit dem USB-Controller auch über diese DLL, die im Programmpaket enthalten ist.

Transfer Protocoll UCP-Controller

Read Buffer

```
Byte0          intern Port1
Byte1          intern Port2
Byte2          intern Counter
Byte3          Key Code, Bit0..3 - Column, Bit4..7 - Row (Zx/Sx)
Byte4          Impulses since last transfer, if 0xFF the key is still pressed
               (if a key is pressed a long time you will get in the first request
                a 0x01, and than 0xFF the next times)
Byte5          Input Port Address 0
Byte6          Input Port Address 1
Byte7          Input Port Address 2
Byte8          Input Port Address 3
Byte9          EEPROM-Variante
```

Write Buffer

```
Byte0          Output Port Address 4
Byte1          Output Port Address 5
Byte2          Output Port Address 6
Byte3          Output Port Address 7
```

Programm-Interface

Funktionen

`int Init_TCP(index)` - Die Funktion öffnet die Kommunikationsschnittstelle und liefert die Versionsnummer (default 0x3000) zurück, der Index ist 0 außer bei Spezialprogrammierungen z.B. bei der Nutzung mehrerer Controller.

`void Close_TCP()` - Die Funktion schließt die Kommunikationsschnittstelle

`READ_BUFFER* Get_TCPKey()` - Die Funktion fragt den Controller ab und schreibt die Daten in einen Lese-Puffer

`READ_BUFFER* TCPTransfer(WRITE_BUFFER*)` - Die Funktion überträgt Daten, die im Schreibpuffer abgelegt sind zum USB-Controller und schreibt die Abfrage-Daten in den Lese-Puffer

`char* Get_TCPDeviceName()` liest den Device-Namen aus

Schnittstelle zu C-Programmen

Header-File

header.h

```
typedef struct _USB_READ_BUFFER {
    BOOLEAN TransferResult;
    UCHAR ReadBuffer[16];
} READ_BUFFER;

typedef UCHAR WRITE_BUFFER[128];

typedef int(WINAPI* cfunc1)(int);
typedef void(WINAPI* cfunc2)();
typedef READ_BUFFER*(WINAPI* cfunc3)();
typedef READ_BUFFER*(WINAPI* cfunc4)(WRITE_BUFFER*);
typedef char*(WINAPI* cfunc5)();
```

Funktions-Aufruf

main.cpp

```
HINSTANCE hLib;
cfunc1 Init_TCP;
cfunc2 Close_TCP;
cfunc3 Get_TCPKey;
cfunc4 TCPTransfer;
cfunc5 Get_TCPDeviceName;
char mod[150];
int i;

UCHAR inBuffer[16];
READ_BUFFER* pinBuffer;
WRITE_BUFFER outBuffer;

boolean load_DLL() {
  BOOLEAN dll_ok;

  hLib = LoadLibrary("ITRA_DEVCOMM.dll");
  if (hLib != NULL) {
    dll_ok = TRUE;
    if (GetModuleFileName((HMODULE)hLib, (LPTSTR)mod, 150) == 0) dll_ok = FALSE;
    Init_TCP = (cfunc1)GetProcAddress((HMODULE)hLib,"Init_TCP");
    Close_TCP = (cfunc2)GetProcAddress((HMODULE)hLib,"Close_TCP");
    Get_TCPKey = (cfunc3)GetProcAddress((HMODULE)hLib,"Get_TCPKey");
    TCPTransfer = (cfunc4)GetProcAddress((HMODULE)hLib,"TCPTransfer");
    Get_TCPDeviceName =
(cfunc5)GetProcAddress((HMODULE)hLib,"Get_TCPDeviceName");
  } else {
    dll_ok = FALSE;
  }
  if (Init_TCP == NULL) dll_ok = FALSE;
```

```
  if (Close_TCP == NULL) dll_ok = FALSE;
  if (Get_TCPKey == NULL) dll_ok = FALSE;
  if (TCPTransfer == NULL) dll_ok = FALSE;
  if (Get_TCPDeviceName == NULL) dll_ok = FALSE;

  return dll_ok;
}

...
  if (load_DLL() == TRUE) {                          // check the DLL
    if ((Init_TCP(0) & 0xFF00) == 0x3000) {          // check the usb driver, 0 =
TCP-Controller

...

      // Code request and LED output
      // fill the writeBuffer with LED vars
      outBuffer[0] = ~(outBuffer[0]);
      outBuffer[1] = ~(outBuffer[1]);
      outBuffer[2] = ~(outBuffer[2]);
      outBuffer[3] = ~(outBuffer[3]);

      pinBuffer=TCPTransfer(&outBuffer);

/*
      // only Code request
      pinBuffer=Get_TCPKey();

*/

      if (pinBuffer->TransferResult == TRUE) {
        for (i=0;i<16;i++) {
          inBuffer[i]=pinBuffer->ReadBuffer[i];
        }
        // process the key events in inBuffer[3], inBuffer[4]

      } else {
        // Transfer Error
      }

      // the Module can be leaved open for loops, but close it on program end
      Close_TCP();
    } else {
      // Error Init
    }
  }
```

Schnittstelle zu Delphi

Definitionen

defs.pas

```
TReadBuffer = record
   KeyResult:boolean;
   Buffer:array[0..15] of byte;
```

```
end;
PReadBuffer = ^TReadBuffer;

TWriteBuffer=array[0..127] of byte;
PWriteBuffer=^TWriteBuffer;

function  Init_TCP(c_nr:integer):integer; stdcall; external 'itra_devcomm.dll';
function  Get_TCPKey:PReadBuffer; stdcall; external 'itra_devcomm.dll';
function  Get_TCPDeviceName:PChar; stdcall; external 'itra_devcomm.dll';
function  TCPTransfer(PWriteB:PWriteBuffer):PReadBuffer; stdcall; external
'itra_devcomm.dll';
procedure  Close_TCP; stdcall; external 'itra_devcomm.dll';

var
  PKeyB:PReadBuffer;
  WriteB:TWriteBuffer;
```

Funktionsaufrufe

Matrix und Port-Abfrage/-Ausgabe

request.pas

```
  // fill the writeBuffer with LED vars
  WriteB[0]:=$FF;
  WriteB[1]:=$FF;
  WriteB[2]:=$FF;
  WriteB[3]:=$FF;

(*
     // only Code request
     PKeyB := Get_TCPKey;
*)

  if Init_TCP(c_nr) = (VersionID + c_nr )then begin    // c_nr is Controller
index
     PKeyB := TCPTransfer(@WriteB);
     if PKeyB^.KeyResult then begin
        // process the key events in Buffer[3], Buffer[4]
        if PKeyB^.Buffer[4] >= 1 then begin
          bKeyCode := PKeyB^.Buffer[3];        // Matrix KeyCode
        end;
        // process switches
        bSwitch0 := PKeyB^.Buffer[5];        // Input-Port0
        bSwitch1 := PKeyB^.Buffer[6];        // Input-Port1
        bSwitch2 := PKeyB^.Buffer[7];        // Input-Port2
        bSwitch3 := PKeyB^.Buffer[8];        // Input-Port3
     end;
  end else begin
    // transfer error
  end;
  // the Module can be leaved open for loops, but close it on program end
  Close_TCP;
```