Navstack / Modules

Mit der Kommunikations-DLL können von jedem beliebigen Programm aus die Ereignisse an der Tastatur-/Encoder-Matrix 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 LED-Anzeigen gefüllt. Im Lesepuffer werden u.a. die Daten der Tastaturmatrix zurückgeliefert.

Transfer Protocoll

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          not used
Byte6          not used
Byte7          not used
Byte8          not used
Byte9          EEPROM-Variante
```

Event-Matrix

```
-------------------------------------------------------------------------------------------------
-----------------------------------------------
     | S0    | S1    | S2    | S3    | S4    | S5    | S6    | S7    | S8    | S9    |
S10(A)| S11(B)| S12(C)| S13(D)| S14(E)| S15(F)|
-------------------------------------------------------------------------------------------------
-----------------------------------------------
Z0   | COMM1 | COMM1 | COMM1 | COMM1 | COMM1 | COMM1 | COMM1 |       | NAV1  | NAV1  |
NAV1  | NAV1  | NAV1  | NAV1  | NAV1  | INTEN-|
     | Key1  | Key2  | Key3  | Key4  | PushB | Down  | Up    |       | Key1  | Key2  |
Key3  | Key4  | PushB | Down  | Up    | SITY  |
     | n.c   | TRANS | SWAP  | RECV  |       |       |       |       | n.c   | IDENT |
SWAP  | MODE  |       |       |       |       |
-------------------------------------------------------------------------------------------------
-----------------------------------------------
Z1   | COMM2 | COMM2 | COMM2 | COMM2 | COMM2 | COMM2 | COMM2 |       | NAV2  | NAV2  |
NAV2  | NAV2  | NAV2  | NAV2  | NAV2  |       |
     | Key1  | Key2  | Key3  | Key4  | PushB | Down  | Up    |       | Key1  | Key2  |
Key3  | Key4  | PushB | Down  | Up    |       |
     | n.c   | TRANS | SWAP  | RECV  |       |       |       |       | n.c   | IDENT |
SWAP  | MODE  |       |       |       |       |
-------------------------------------------------------------------------------------------------
-----------------------------------------------
Z2   | ADF   | ADF   | ADF   | ADF   | ADF   | ADF   | ADF   |       | XPDR  | XPDR  |
XPDR  | XPDR  | XPDR  | XPDR  | XPDR  |       |
     | Key1  | Key2  | Key3  | Key4  | PushB | Down  | Up    |       | Key1  | Key2  |
Key3  | Key4  | PushB | Down  | Up    |       |
```

```
     | n.c   | IDENT | SWAP  | MODE  |       |       |       | n.c   | n.c.  |
n.c   | MODE  |       |       |       |       |
-------------------------------------------------------------------------------------
-----------------------------------------------
Z3   | AP    | AP    | AP    | AP    | AP    | AP    | AP    | AP    | AP    |
AP    | AP    | AP    | AP    | AP    | AP    |
     | ALT   | ALT   | ALT   | ALT   | VS    | VS    | VS    | VS    | APPR  | FD    |
MASTER| n.c   | NAV   | Y/D   | LVL   | LOC   |
     | PushB | Down  | Up    | Key   | PushB | Down  | Up    | Key   |       |       |
|       |       |       |       |       |       |
-------------------------------------------------------------------------------------
-----------------------------------------------
Z4   |       | DME   |       | DME   | DME   | DME   | DME   | DME   | AP    | AP    |
AP    | AP    |       |       |       |       |
     |       | Key2  |       | Key4  | PushB | Down  | Up    | Key   | HDG   | HDG   |
HDG   | HDG   |       |       |       |       |
     |       | IDENT |       | 1/2   | n.c   | n.c   | n.c   | n.c   | PushB | Down  |
Up    | Key   |       |       |       |       |
-------------------------------------------------------------------------------------
-----------------------------------------------
Z5   | RMP   | RMP   | RMP   | RMP   | RMP   | RMP   | RMP   |       | MFP   | MFP   |
MFP   | MFP   | MFP   | MFP   | MFP   |       |
     | Key1  | Key2  | Key3  | Key4  | PushB | Down  | Up    |       | Key1  | Key2  |
Key3  | Key4  | PushB | Down  | Up    |       |
     | MODE  | COMM1 | SWAP  | COMM2 |       |       |       |       | MODE  | COMM1 |
SWAP  | COMM2 |       |       |       |       |
-------------------------------------------------------------------------------------
-----------------------------------------------
Z6   | RMP   | RMP   | RMP   | RMP   |       |       |       |       | MFP   | MFP   |
MFP   | MFP   |       |       |       |       |
     | Key1  | Key2  | Key3  | Key4  |       |       |       |       | Key1  | Key2  |
Key3  | Key4  |       |       |       |       |
     | NAV1  | NAV2  | ADFP  | XPDR  |       |       |       |       | NAV1  | NAV2  |
QNH   | SPEED |       |       |       |       |
-------------------------------------------------------------------------------------
-----------------------------------------------
```

Write Buffer

```
Byte0          Port0, OMI LEDs NAVSTACK (low active 0-ON, 1-OFF)
                  bit4 Inner marker
                  bit2 Middle marker
                  bit0 Outer marker
Byte1          not used
Byte2          not used
Byte3          not used

Byte4          Key LEDs AUTOPILOT (high active 1-ON, 0-OFF)
                  bit7 NAV
                  bit6 Y/D
                  bit5 LVL
                  bit4 LOC or BC
                  bit3 APR
                  bit2 FD
                  bit1 AP
                  bit0 frei

Byte5          Key LEDs RMP/MFP-MODE (high active 1-ON, 0-OFF)
```

```
                  bit7 MFP Speed
                  bit6 MFP QNH
                  bit5 MFP NAV2
                  bit4 MFP NAV1

                  bit3 RMP XPDR
                  bit2 RMP ADF
                  bit1 RMP NAV2
                  bit0 RMP NAV1

Byte6..9, 10..13 Display0 (COMM1 or RMP with address 0)
               Byte6
                  bit0..3 Digit0 Active freq
                  bit4..7 Digit1 Active freq
               Byte7
                  bit0..3 Digit2 Active freq
                  bit4..7 Digit3 Active freq
               Byte8
                  bit0..3 Digit4 Active freq
                  bit4..7 four single LEDs,
                          only bit4 is used for Transmit/Ident signalisation
                          leave other bits 0
               Byte9 Decimal Point
                  bit5 Digit4
                  bit4 Digit3
                  bit3 Digit2
                  bit2 Digit1
                  bit1 Digit0
                  bit0 without
               -------------------
               Byte10
                  bit0..3 Digit0 Standby freq
                  bit4..7 Digit1 Standby freq
               Byte11
                  bit0..3 Digit2 Standby freq
                  bit4..7 Digit3 Standby freq
               Byte12
                  bit0..3 Digit4 Standby freq
                  bit4..7 four Key LEDs RMP/MFP-MODE ,
                          only bit4,5,7 used for MODE, COMM1, COMM2 Keys
                          leave 0 if not used
               Byte13 Decimal Point
                  bit5 Digit4
                  bit4 Digit3
                  bit3 Digit2
                  bit2 Digit1
                  bit1 Digit0
                  bit0 without

Byte14..21     Display1 (COMM2 or MFP with address 1)
Byte22..29     Display2 (NAV1)
Byte30..37     Display3 (NAV2)

Byte38         0 - all LEDs normal operation, 0x01 - all LEDs shutdown
Byte39         Intensity: bit0..3 for 7segment LEDs (please don't set over 0x03 because
of current)
                          bit4..7 for single LEDs   (please don't set over 0x60 because
of current)
                          example: BYTE intensity[4]= {0x10,0x21,0x42,0x63};
```

```
Byte40          RMP display address, possible values are 0, 4, 8; default is 4
Byte41          MFP display address, possible values are 1, 5, 9; default is 5

Byte64..71      Display4 (ADF or RMP with address 4 - default in MODULES)
Byte72..79      Display5 (XPDR or MFP with address 5 - default in MODULES)
Byte80..87      Display6 DME
Byte88..95      Display7 AP
Byte96..103     Display8 RMP with address 8 - default in NAVSTACK (this address is set
to use COMM1/2, NAV1/2, ADF, XPDR in the NAVSTACK)
Byte104..111    Display9 MFP with address 9 - default in NAVSTACK (this address is set
to use COMM1/2, NAV1/2, ADF, XPDR in the NAVSTACK)
```

Segment-Decoder for BCD

```
0 - number 0
1 - number 1
2 - number 2
3 - number 3
4 - number 4
5 - number 5
6 - number 6
7 - number 7
8 - number 8
9 - number 9
A - letter A
B - letter H
C - lower segment
D - upper segment
E - middle segment
F - blank
```

Programm-Interface

Funktionen

`int Init_MODULES` - Die Funktion öffnet die Kommunikationsschnittstelle und liefert die Versionsnummer (default 0x2200) zurück.

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

`READ_BUFFER* MODULESTransfer(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_MODULESDeviceName()` 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* cfunc4)(WRITE_BUFFER*);
typedef char*(WINAPI* cfunc5)();
```

Funktions-Aufruf

main.cpp

```
HINSTANCE hLib;
cfunc1 Init_MODULES;
cfunc2 Close_MODULES;
cfunc4 MODULESTransfer;
cfunc5 GetMODULESDeviceName;
char mod[150];
int i;

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

boolean load_DLL() {
  BOOLEAN dll_ok;

  hLib = LoadLibrary("ITRA_APCOMM.dll");
  if (hLib != NULL) {
    dll_ok = TRUE;
    if (GetModuleFileName((HMODULE)hLib, (LPTSTR)mod, 150) == 0) dll_ok = FALSE;
    Init_MODULES = (cfunc1)GetProcAddress((HMODULE)hLib,"Init_MODULES");
    Close_MODULES = (cfunc2)GetProcAddress((HMODULE)hLib,"Close_MODULES");
    MODULESTransfer = (cfunc4)GetProcAddress((HMODULE)hLib,"MODULESTransfer");
    Get_MODULESDeviceName =
(cfunc5)GetProcAddress((HMODULE)hLib,"Get_MODULESDeviceName");
    } else {
      dll_ok = FALSE;
    }
    if (Init_MODULES == NULL) dll_ok = FALSE;
    if (Close_MODULES == NULL) dll_ok = FALSE;
    if (MODULESTransfer == NULL) dll_ok = FALSE;
    if (Get_MODULESDeviceName == NULL) dll_ok = FALSE;
    return dll_ok;
  }

...
  if (load_DLL() == TRUE) {                          // check the DLL
    if ((Init_MODULES(0) & 0xFF00) == 0x2200) {      // check the usb driver
...

      // Code request and LED output
      // fill the writeBuffer with LED vars
      outBuffer[0] = ....;
...
```

```
      pinBuffer=MODULESTransfer(&outBuffer);

      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_MODULES();
    } else {
      // Error Init
    }
  }
```

Schnittstelle zu Delphi

Definitionen

defs.pas

```
TReadBuffer = record
  TransferResult:boolean;
  readBuffer:array[0..15] of byte;
end;
PReadBuffer = ^TReadBuffer;

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

function  Init_MODULES:integer; stdcall; external 'itra_apcomm.dll';
function  MODULESTransfer(PWriteB:PWriteBuffer):PReadBuffer; stdcall; external
'itra_apcomm.dll';
function  Get_MODULESDescr:PUSBDeviceDescriptor; stdcall; external
'itra_apcomm.dll';
procedure Close_MODULES; stdcall; external 'itra_apcomm.dll';

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

Funktionsaufrufe

request.pas

```
  // fill the writeBuffer
  WriteB[0]:=...;
  ...
```

```
if Init_MODULES = (VersionID) then begin
   PKeyB := MODULESTransfer(@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;
   end;
end else begin
  // transfer error
end;
// the Module can be leaved open for loops, but close it on program end
Close_MODULES;
```