Datenaustausch zwischen Roboter und PC via WLAN
Erfahrungsbericht:

Im folgenden Bericht möchte ich auf die Möglichkeit eingehen, wie man Daten zwischen PC und Roboter via WLAN austauschen kann. Meine Wahl fiel auf das Embedded WLAN Modul der Firma Avisaro. Das Modul kann, je nach Auslieferungszustand via CAN Bus, I2C Bus, SPI oder auch über RS232 Schnittstelle mit dem Roboter kommunizieren. Ich habe mich für die Modulversion mit I2C Bus entschieden.

Eingesetzte Software / Hardware

- Roboter RP6 der Firma AREXX (www.arexx.com)

- Borland C++ Compiler auf dem PC (www.borland.com)

- PC OS ist Windows XP Home

- Emulations- Software der Firma HWgroup (www.hw-group.com)

- WLAN Adapter der Firma Avisaro (www.avisaro.de)

- Einfacher WLAN Router von D-Link

Beschreibung der Komponenten:

1) Das WLAN Modul der Firma Avisaro wird über den I2C Bus auf Seiten des Roboters angesprochen, dabei war es mir wichtig, die vorhandenen Routinen zu nutzen, die mit dem RP6 Roboter mit geliefert werden. Das WLAN Modul ist per default als Slave konfiguriert mit der I2C Adresse 73. Somit müssen die I2C Master Routinen auf dem Roboter benutzt werden. Die Frequenz der I2C Busses habe ich auf 100KHz gelassen. Das WLAN Modul kann auch, via Clock-Stretching, mit höherer Frequenz angesprochen werden

2) Die Software HW VSP3 „virtual serial port“ der Firma HW-Group wird auf der PC Seite eingesetzt, um die Kommunikation zur PC WLAN Karte zu vereinfachen.

3) Es ist Möglich eine Peer zu Peer Verbindung zwischen PC und WLAN Modul aufzubauen. Ich habe aber die Lösung via WLAN Router gewählt. Das WLAN Modul ist Standardmäßig auf die IP Adresse 192.168.0.73 konfiguriert. Auf die Konfiguration des WLAN Moduls via Weboberfläche möchte ich hier nicht eingehen und verweise hiermit auf die mitgelieferte Dokumentation.

Wichtig sind nur folgende Parameter:

- Netzwerksetup als TCP Server

- I2C-Bus No ACK

Funktionsbeschreibung:

PC Seite:

Serielle Schnittstelle / Emulations-Software

Das PC Programm schreibt / liest Daten von und zu einer seriellen Schnittstelle und nicht direkt auf den PC WLAN Adapter. Die serielle Schnittstelle wird durch die HW VSP3 Software emuliert und übernimmt Schreiben und Lesen der Daten auf den PC WLAN Adapter. Somit vereinfacht sich die Kommunikation von und zum WLAN Adapter wesentlich, da sich die Kommunikation auf Lesen und Schreiben zur serielle Schnittstelle beschränkt. Hier zu gibt es jede Menge Beispiele im Netz, oder auf der MSDN Webseite bei Microsoft, wie eine serielle Schnittstelle unter Windows XP angesprochen wird. Ist das WLAN Modul auf dem Roboter im Netzwerk mit seiner IP Nummer erkannt worden, dann kann via der Emulationssoftware ein COM Port zugewiesen werden z.B. COM3.

Die Standardmäßige Einstellung der seriellen Schnittstelle ist, 9600 Baud, 8 Datenbits, 1 Stopbit, No Parity. Es ist an dieser Stelle sehr hilfreich sich das Logfile der Emulations Software anzusehen, da die Parameter der seriellen Schnittestelle herausgeschrieben werden und man sofort erkennt, ob das Modul erfolgreich connected wurde.

Programm auf dem PC Zur Erstellung der PC seitigen Software habe ich Borland C++ benutzt. Die Routinen im Sourcecode sind wie bereits beschrieben von MSDN kopiert und angepasst worden. Ich möchte hier nicht so tief auf die Beschreibung des Codes eingehen, dass kann man im angehängten SourceCode viel besser sehen. Das Programm besteht eigentlich nur aus einer lesenden Routine, die Daten aus der seriellen Schnittstelle ausliest und auf dem Bildschirm in einer DOS-Box darstellt. Senden von Daten an den Roboter ist auch möglich und als Funktion hinterlegt.

Netzwerk:

WLAN:

Das WLAN-Netz wurde von mir mit den Standardparametern aufgesetzt. Ich habe nur die IP Adresse, die SSID und den Kanal angepasst. WEP / WAP Verschlüsselung ist möglich und kann via Webpage auf dem WLAN Modul eingestellt werden (siehe Dokumentation)

Roboter Seite:

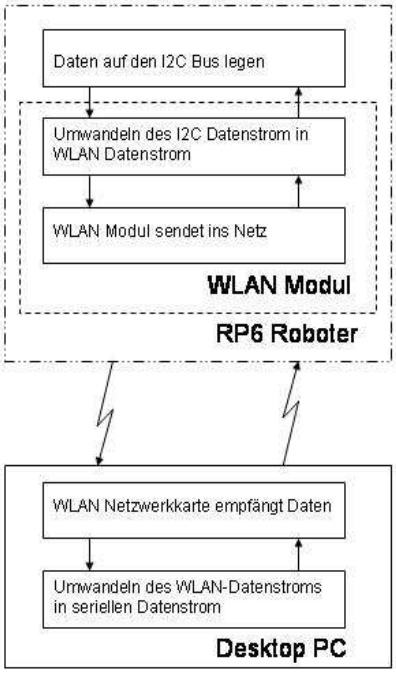
WLAN Modul:

Das WLAN Modul wird über den I2C Bus angesprochen, die I2C Adresse habe ich behalten, da die Adresse 73 von keinem anderen I2C Device genutzt wird. Wichtig war für mich die fertigen Routinen, die mit den RP6 Libraries geliefert werden, zu benutzen. Dadurch musste ich keine extra Funktionen entwickeln die den Lese / Schreibzugriff auf den I2C Bus

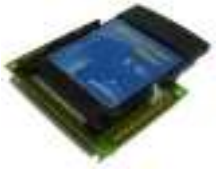
ausführen. Allerdings ist zu beachten, dass bei schreibenden Zugriffen auf das WLAN Modul die I2C Adresse um ein Bit verschoben werden muss. Das Bit 0 muss mit einer Null gefüllt werden (Adr<<1)

Schwieriger ist der lesenden Zugriff auf das Modul, dazu muss die Adresse ebenfalls um ein Bit verschoben werden und das Bit 0 auf EINS gesetzt werden (Adr<<1 | 1). Wichtig: Im Normalfall adressiert man, als ersten Schritt, ein I2C Device und das zu lesende Register von dem man die Daten abholen möchte. Anschließend greift man lesend auf das zuvor adressierte Register des I2C Devices zu. Will man lesend auf das WLAN Modul zugreifen, so habe ich festgestellt, dass man direkt lesend auf die I2C Adresse zugreifen kann, ohne das Device vorher, durch einen Schreibzugriff , zu adressieren. Die ersten zwei Byte, der abgeholten Daten, sind die Länge / Size der auf dem WLAN Modul bereitgestellten Daten. Somit kann man einfach feststellen wie viele Daten aus dem Device ausgelesen werden müssen. Alle Daten müssen aus dem Device abgeholt werden. Siehe auch SourceCode Beispiele. Die verwendeten Software-Routinen werden mit dem RP6 Roboter mitgeliefert. Ebenso der verwendete AVR Compiler und dessen Libraries.

Blockdiagramm



LAN Karte von Avisaro



Sourcecode für den PC:

```
#include <windows.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <commctrl.h>



//  ********  Globale Variablen  ********
char          *gszPort;
HANDLE        hCom = NULL;
DCB           dcb;
COMMTIMEOUTS  ct;
bool          CommFlag;
bool          ReadOnly;
char          string[100];
BYTE          data_in[100], data_out, buff[100];
int           k, m;
DWORD         n;



// ************* COM Port initialisieren **************
int setup_Com_Port ()
{
   gszPort = string;
   hCom = CreateFile(gszPort, GENERIC_READ | GENERIC_WRITE, 0, 0,
OPEN_EXISTING, 0, 0);
   if (hCom == INVALID_HANDLE_VALUE)
   {
       printf ("\n\n \a Error Initialise Port");
       CommFlag = false;
       return 0;
   }
   dcb.DCBlength = sizeof(DCB);
   GetCommState(hCom, &dcb);
   if (!BuildCommDCB("baud=9600 parity=N data=8 stop=1", &dcb))
   {
       hCom = NULL;
       printf ("\n\n \a Error kann Portparameter nicht setzen e.g. 9600
Boud");
       CommFlag = false;
   }
   if(!SetCommState(hCom, &dcb))
   {
       CloseHandle(hCom);
       hCom = NULL;
       printf ("\n\n \a Cannot set comm state  BCD.\n");
       CommFlag = false;
       return 0;
   }
   ct.ReadIntervalTimeout         = MAXDWORD;
   ct.ReadTotalTimeoutConstant    = 0;
   ct.ReadTotalTimeoutMultiplier  = 1;
   ct.WriteTotalTimeoutConstant   = 250;
   ct.WriteTotalTimeoutMultiplier = 1;
```

```
   if(!SetCommTimeouts(hCom, &ct))
   {
       CloseHandle(hCom);
       hCom = NULL;
       printf ("\n \a Cannot set comm timeouts.\n");
       CommFlag = false;
        return 0;
   }
   return 0;
}
```

Senden von Daten an die serielle Schnittstelle

```
/* ********** Senden der Daten an Serielle Schnittstelle ***** */
void sende_cmd_start (BYTE cmd[10], int howmuch)     // Datenbytes und
Anzahl wieviele  Bytes   gesendet werden sollen
{
       DWORD  n;
       WriteFile(hCom, &cmd, howmuch, &n, NULL);
}
```

Von der serielle Schnittstelle Daten auslesen

```
/* **** Endlos Schleife des Hauptprogramms **** */
if (hCom != NULL)
{
 while(CommFlag)                          // Bearbeiten solange das COM
Port offen ist
 {
    ReadFile(hCom, &cmd, 9, &n, NULL); // Daten aus der Schnittstelle
auslesen, hier 9 Bytes
    if(n)                     // Wenn Daten ausgelesen wurden, sichern im
Feld  data_in
    {                         // Daten gesichert für weitere Verarbeitung
        data_in[0] = cmd[0];
        data_in[1] = cmd[1];
        data_in[2] = cmd[2];
        data_in[3] = cmd[3];
        data_in[4] = cmd[4];
        data_in[5] = cmd[5];
        data_in[6] = cmd[6];
        data_in[7] = cmd[7];
        data_in[8] = cmd[8];
    }
 }
}
```

Beim verlassen des Programms serielle Schnittstelle schließen

```
/*   ************ Port schliessen ******* */
  Sleep (5);
  CloseHandle(hCom);
  hCom = NULL;
  gotoxy (2,32);
  printf ("\n  Port %s geschlossen ", string);
  printf ("\n\n Programm beendet ");
  return 0;
```

Sourcecode für den RP6 Roboter:

Auszug aus dem Hauptprogramm:

```
// ISR Handler initialisieren
I2CTWI_initMaster(100);      // Initialize the TWI Module for Master
operation with 100kHz  SCL Frequency
I2CTWI_setTransmissionErrorHandler(I2C_transmissionError);   // Register
the event hand



// ********************** Main loop ***********************************
while(true)
{
  behaviourController();              // Aufruf der Haupt-Controll-Funktion
  task_RP6System();                   // Abfrage des ADC, ACS, IRCOMM
usw...
  if (getStopwatch1() > 200)
  {
      task_commandProcessor();        // Abfrage ob ein Commando gesendet
wurde
      setStopwatch1(0);
  }
  task_I2CTWI();                // Call I2C Management routine
}
return 0;
```

Lesen von Daten von der WLAN Karte bereitgestellt über den I2C Bus

```
// **** Comandos von PC über WLAN Karte einlesen
void task_commandProcessor(void)
{
  uint16_t     size;
  uint8_t      messageBuf [100];
  size = 0;
  if(!I2CTWI_isBusy())
  {
     I2CTWI_readBytes (73<<1 | 1, &messageBuf[0],  38);      // Daten von
WLAN Karte einlesen. Hier werden  einfach mal 38 Byte eingelesen
     size = messageBuf[0]<<8 | messageBuf[1];  // Ermitteln wie viel
tatsächlich Byte gesendet                          wurden
  }                                          // Diese Funktion muss noch
optimiert werden
}
//  In dem Feld messageBuf [2]  bis  messageBuf[size]  befinden sich die
Daten von PC gesendet
```

Senden von Daten an die WLAN Karte über den I2C Bus

```
// Mit folgender Routine können eine belibige Anzahl, hier 9 Byte, an die
WLAN Karte gesendet werden
// Daten im Feld „buffer“ zur WLAN Karte senden via  I2C  Bus
if (!I2CTWI_isBusy ())   // Daten über I2C an WLAN senden
   I2CTWI_transmitBytes (73<<1, &buffer[0], 9);
```

Dipl. –Ing.Peter Schneider

Disclaimer