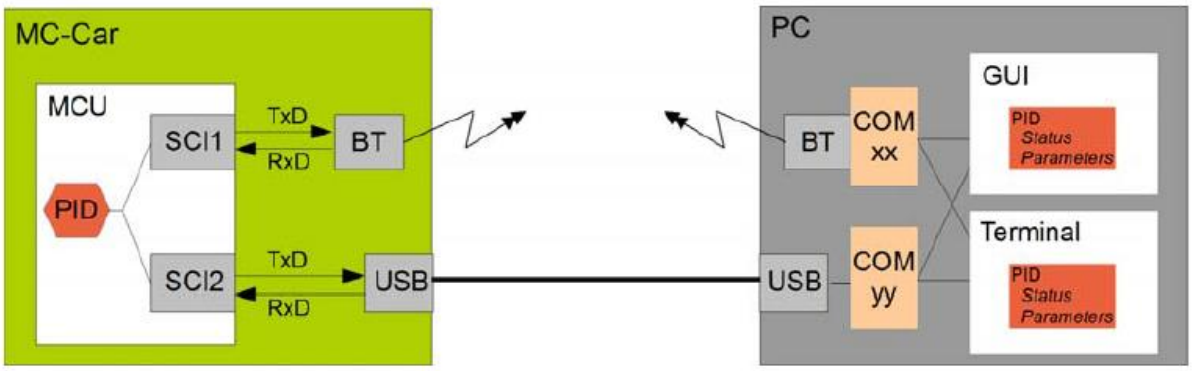
10: Serial Communication Interface (SCI)

Sie verstehen das RS-232 Protokoll sowie das Zusammenspiel zwischen HW und SW bei der Kommunikation über die serielle Schnittstelle.

1. Systemüberblick

Vom PC aus sollen im MC-Car die Parameter des PID-Geschwindigkeitsreglers eingestellt sowie der aktuelle Status (Sollwert, Istwert, Regelabweichung etc.) gelesen werden können.

Nachfolgend ist der Systemaufbau für die serielle Kommunikation zwischen MC-Car und PC gezeigt. Die beiden möglichen physischen Verbindungen (USB, Bluetooth) sowie die beiden möglichen Benutzerschnittstellen (Terminal, GUI) sind in den folgenden Kapiteln beschrieben.



2. Physische Verbindungen

a) Verbindung über USB

Das Debug-Interface auf dem MC-Car beinhaltet zusätzlich einen USB-zu-RS232 Wandler. Im Geräte-Manager (unter Systemsteuerung => System) sollten Sie den entsprechenden Com-Port ausfindig machen können. Es wird kein zusätzlicher Treiber benötigt.

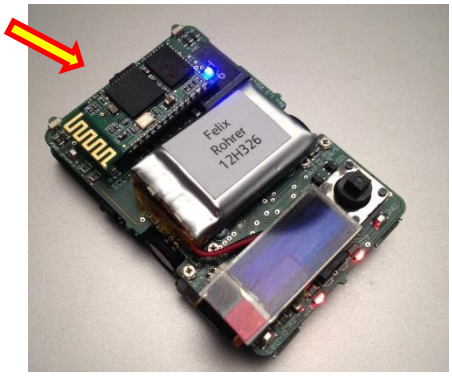
Um über USB zu kommunizieren, stellen Sie in sci.h die Konfiguration um auf SCI2:

```
#ifndef SCI_H
#define SCI_H

#include "platform.h"

// 1 = Bluetooth, 2 = Cable
#define SCI_DEFAULT                2       // [0|1|2]  0=>aus, 1=>SCI1, 2=>SCI2
```

b) Verbindung über Bluetooth

Bei dieser Variante müssen sie zuerst das Bluetooth-Modul bei ausgeschaltetem MC-Car aufsetzen. Achten sie dabei unbedingt auf die Polarität und die richtige Positionierung der Steckerreihen übereinander.

Achtung: Das Bluetooth-Modul kann durch falsches Aufstecken zerstört werden!



Wählen Sie als nächstes die Seriennummer als Namen für das Bluetooth Modul und tragen Sie diese in platform.h ein:

```
#define BLUETOOTH_NAME          "12H301"
```

Um über Bluetooth zu kommunizieren, stellen Sie in sci.h die Konfiguration um auf SCI1:

```
#ifndef SCI_H
#define SCI_H

#include "platform.h"

// 1 = Bluetooth, 2 = Cable
#define SCI_DEFAULT                1       // [0|1|2]  0=>aus, 1=>SCI1, 2=>SCI2
```

Stellen Sie nun wie folgt eine Verbindung zum Bluetooth Modul her:

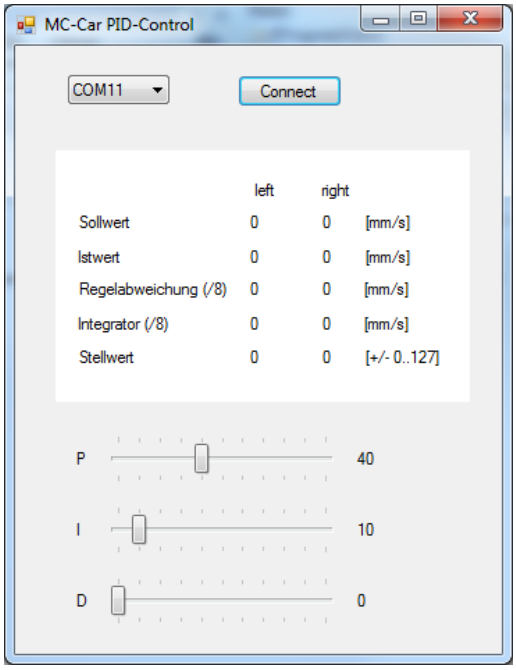
(Win7) Start => Geräte und Drucker => Geräte hinzufügen

Hier sollten Sie Ihren MC-Car sehen. Das Passwort zum Verbindungsaufbau lautet: „**1234**“

3. Benutzerschnittstellen

a) GUI

Starten Sie die Applikation MCCar.exe und verbinden Sie sich über den entsprechenden COM-Port mit dem MC-Car.



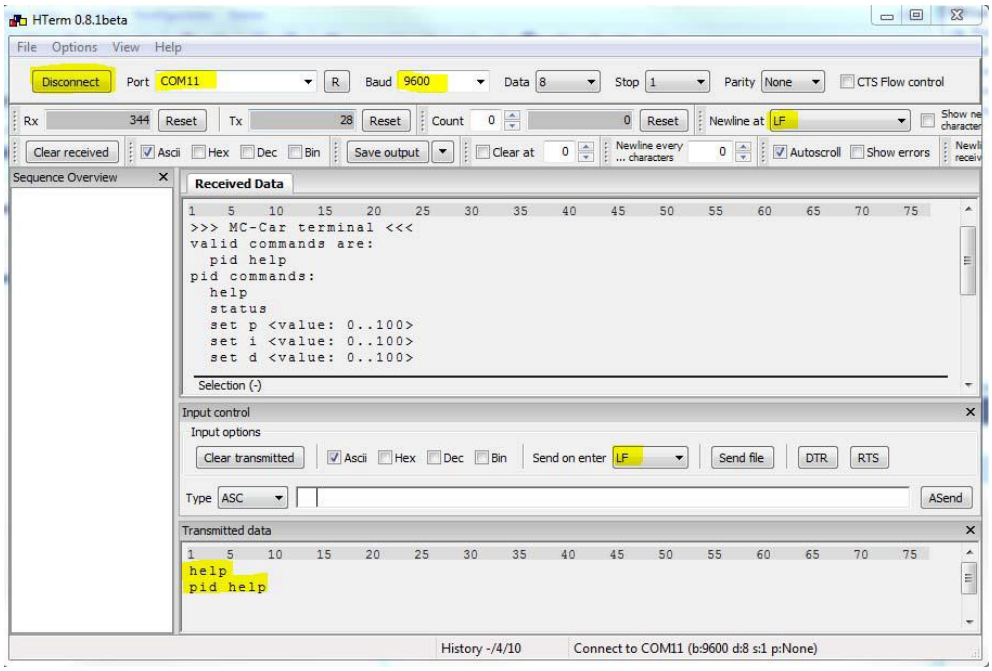
b) Terminalprogramm

Nehmen Sie im Terminalprogramm (z.B. HTerm) die unten gezeigten Einstellungen vor:

Baudrate = 9600, Format = 8-None-1, Newline at = LF, Send on Enter = LF

und verbinden Sie sich über den entsprechenden COM-Port mit dem MC-Car.

Nachdem Senden des Kommandos „help“ vom Terminalprogramm, werden die verfügbaren Kommandos angezeigt, siehe Implementierung in term.c.



4. RS-232 Software für MC-Car

Legen Sie in CW ein neues C-Projekt mit Copy/Paste an und verwenden Sie die gegebenen Files main.c und Project.prm für das neue Projekt. Aktualisieren Sie in Ihrer Bibliothek MC_Library die Verzeichnisse Lib_Headers bzw. Lib_Sources mit den gegebenen Bibliotheksdateien.

Analysieren Sie den gegebenen Code und implementieren Sie im File sci1_temp.c oder/und sci2_temp.c (je nachdem ob Sie mit Bluetooth oder/und USB-Verbindung arbeiten) die folgenden Funktionalitäten (siehe CW Task-View):

a) in sci1/2Init() eine Baud Rate von 9600 Baud/s unter Verwendung des define-Wertes BUSCLOCK.
   Stellen Sie dabei sicher, dass der zu konfigurierende SBR -Wert zum nächsten ganzzahligen Wert gerundet wird.
   Tipp: x + 0.5 = (10 x + 5) / 10
b) in der ISR sci1/2RxD() das Auslesen der empfangenen Bytes aus dem SCI Data Register und Schreiben in den Rx-Ringbuffer. Danach sollten die Reglerparameter z.B. über das GUI im MC-Car verändert werden können.
c) in der ISR sciXTxD() das Auslesen der zusendenden Bytes aus dem Tx-Rinbuffer und Schreiben ins SCI Data Register. Danach sollte ebenfalls das Auslesen des Regler-Status funktionieren.

sci1.c

[…]

```
/**
 * RxD Interrupt-Routine
 *
 * Liest das empfangene Byte und speichert es in der Empfangsqueue.
 *
 * @details
 * Um den Empfangsinterrupt zu quittieren muss zuerst da Statusregister SCIxS1 gelesen
 * werden und anschliessend muss das empfange Zeichen aus SCIxD gelesen werden.
 */
interrupt void sci1RxD(void)
{
  char ch;
  (void)SCI1S1;       // 1. Status lesen
  ch = SCI1D;         // 2. empf. Zeichen lesen => Interrupt wird quittiert.

  // @ToDo write received data byte into the rx buffer
  if (rx1BufCount < SCI1_RX_BUF_SIZE)
  {
    rx1Buf[rx1BufWritePos] = ch;
    rx1BufCount++;
    rx1BufWritePos++;
    if (rx1BufWritePos == SCI1_RX_BUF_SIZE) rx1BufWritePos = 0;
  }
}


/**
 * TxD Interrupt-Routine
 *
 * Holt das nächste Byte aus der Queue und versendet es.
 * Falls die Queue leer ist, dann wird der Interrupt deaktiviert.
 */
interrupt void sci1TxD()
{
  (void)SCI1S1;       // 1. Status lesen

  // @ToDo read from the tx buffer and write the data into the sci data register.
  // Disable tx interrupt if tx buffer is empty
  if (tx1BufCount > 0)
  {
    SCI1D = tx1Buf[tx1BufReadPos];
    tx1BufCount--;
    tx1BufReadPos++;
    if (tx1BufReadPos == SCI1_TX_BUF_SIZE) tx1BufReadPos = 0;
  } else {
    // Buffer leer => Transmit Data Register Empty Interrupt deaktivieren
    // wird später durch sciWriteByte() wieder aktiviert...
    SCI1C2_TIE = 0;
  }
}
```

[…]

sci2.c

[…]

```
/**
 * RxD Interrupt-Routine
 *
 * Liest das empfangene Byte und speichert es in der Empfangsqueue.
 *
 * @details
 * Um den Empfangsinterrupt zu quittieren muss zuerst da Statusregister SCIxS1 gelesen
 * werden und anschliessend muss das empfange Zeichen aus SCIxD gelesen werden.
 */
interrupt void sci2RxD(void)
{
  char ch;
  (void)SCI2S1;            // 1. Status lesen
  ch = SCI2D;        // 2. empf. Zeichen lesen => Interrupt wird quittiert.

  // @ToDo write received data byte into the rx buffer
  if (rx2BufCount < SCI2_RX_BUF_SIZE)
  {
    rx2Buf[rx2BufWritePos] = ch;
    rx2BufCount++;
    rx2BufWritePos++;
    if (rx2BufWritePos == SCI2_RX_BUF_SIZE) rx2BufWritePos = 0;
  }
}


/**
 * TxD Interrupt-Routine
 *
 * Holt das nächste Byte aus der Queue und versendet es.
 * Falls die Queue leer ist, dann wird der Interrupt deaktiviert.
 */
interrupt void sci2TxD()
{
  (void)SCI2S1;       // 1. Status lesen

  // @ToDo read from the tx buffer and write the data into the sci data register.
  // Disable tx interrupt if tx buffer is empty
  if (tx2BufCount > 0)
  {
    SCI2D = tx2Buf[tx2BufReadPos];
    tx2BufCount--;
    tx2BufReadPos++;
    if (tx2BufReadPos == SCI2_TX_BUF_SIZE) tx2BufReadPos = 0;

  } else {
    // Buffer leer => Transmit Data Register Empty Interrupt deaktivieren
    // wird später durch sciWriteByte() wieder aktiviert...
    SCI2C2_TIE = 0;
  }
}
```

[…]