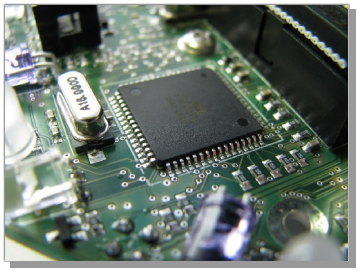
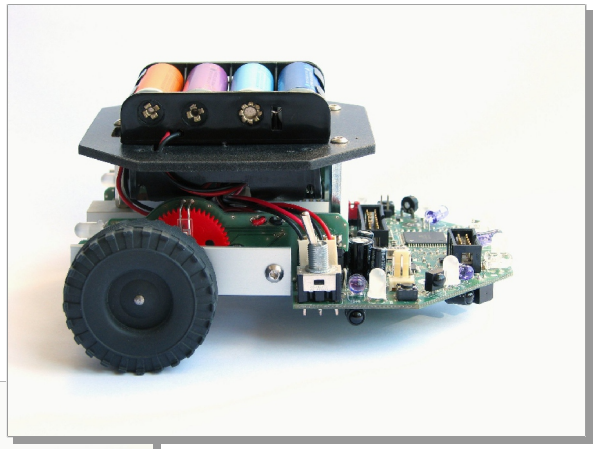
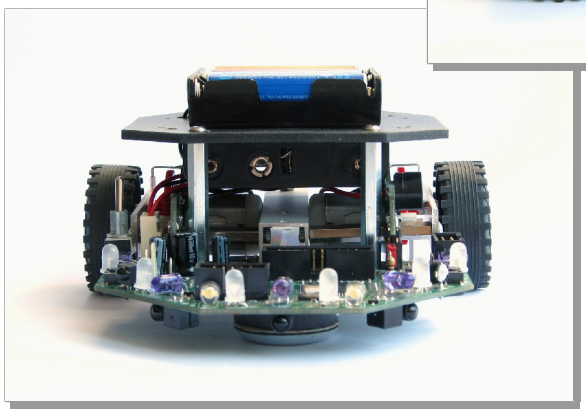
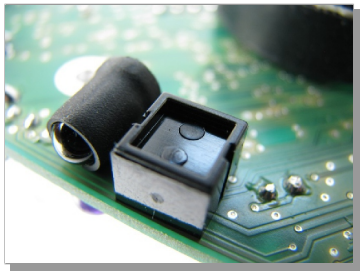
Roboterbausatz Nibo2

Tutorial zur Programmierung









nicai systems

nibo.nicai-systems.de

Version 11.12.2012

Inhaltsverzeichnis

1 Einleitung

Dieses Tutorial bietet eine Schritt für Schritt Anleitung für die erste Inbetriebnahme des Nibo. Mittels kleiner, ausführlich erklärter Beispiele soll der Leser mit der grundlegenden Funktionalität der Kernkomponenten des Roboters vertraut gemacht werden.

Sie lernen in den folgenden Abschnitten wie die Programmierumgebung installiert wird, wie die LEDs zum Blinken/Leuchten gebracht werden können, wie Sie Text und Sensorwerte auf dem Display sichtbar machen können und letztendlich auch, wie Sie den Nibo in Bewegung bringen!

Das Tutorial richtet sich an Robotik-/Programmieranfänger, um ihnen einen leichten Einstieg in die Gebiete der Programmierung, insbesondere der Mikrocontroller-Programmierung zu ermöglichen.

2 Installation der Programmierumgebung

2.1 AVR Studio 4

Zunächst muss das AVR Studio 4.19 (build 730) von Atmel installiert werden. Hierzu laden Sie sich die Installationsdatei von folgender URL herunter, vorab verlangt Atmel von Ihnen eine Registrierung:

http://www.atmel.com/tools/studioarchive.aspx

Doppelklicken Sie nun das heruntergeladene .exe-File und installieren Sie das AVR Studio 4.19 auf Ihrem Computer.

AVR Studio 4.19 ist eine von Atmel kostenlos zur Verfügung gestellte Entwicklungsumgebung (IDE) für AVR-Mikrocontroller mit der Sie Ihre Nibo-Projekte verwalten können.

2.2 WinAVR

Anschließend wird das WinAVR installiert. Hierzu laden Sie die aktuelle Version der Datei **WinAVR-xxx-install.exe** von der Seite http://sourceforge.net/projects/winavr/ herunter. Doppelklicken Sie anschließend auf die Datei und folgen Sie dem Installationsmanager.

WinAVR ist eine Sammlung von vielen wichtigen Softwarepaketen für die AVR-Entwicklung unter Windows. Die Sammlung enthält unter anderem den C/C++ Compiler avr-gcc, die C-Standardbibliothek avr-libc, die „binutils“ und die Programmiersoftware AVRDUDE.

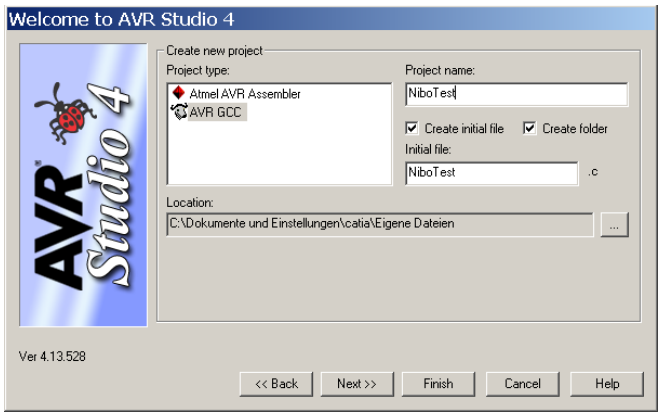
2.3 Nibo Library

Jetzt wird die Nibo Library installiert. Hierzu laden Sie sich den Installer **nibolib-xxx.msi** von der Seite http://sourceforge.net/projects/nibo/ herunter und starten den Installer mit einem Doppelklick.

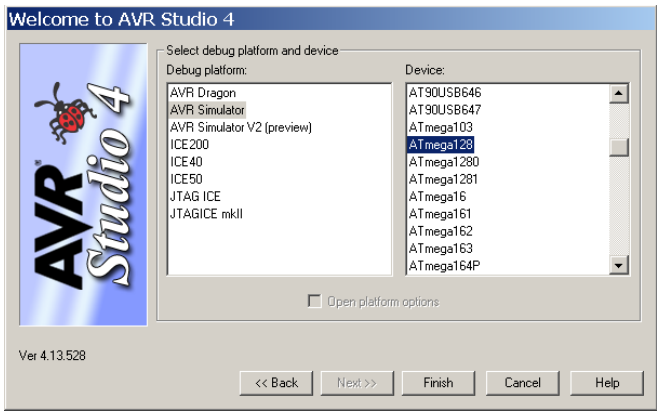
Die NiboLib enthält C und C++ Routinen zur einfachen Ansteuerung des Roboters.

3 Ein erstes Projekt anlegen

Nun soll im AVR Studio ein neues Projekt angelegt werden. Starten Sie dazu das AVR Studio. Klicken Sie im Willkommensbildschirm auf „*New Project*“. Im nun erscheinenden Fenster wählen Sie als *Project type* **AVR GCC** aus und tragen bei *Project name* **NiboTest** ein.

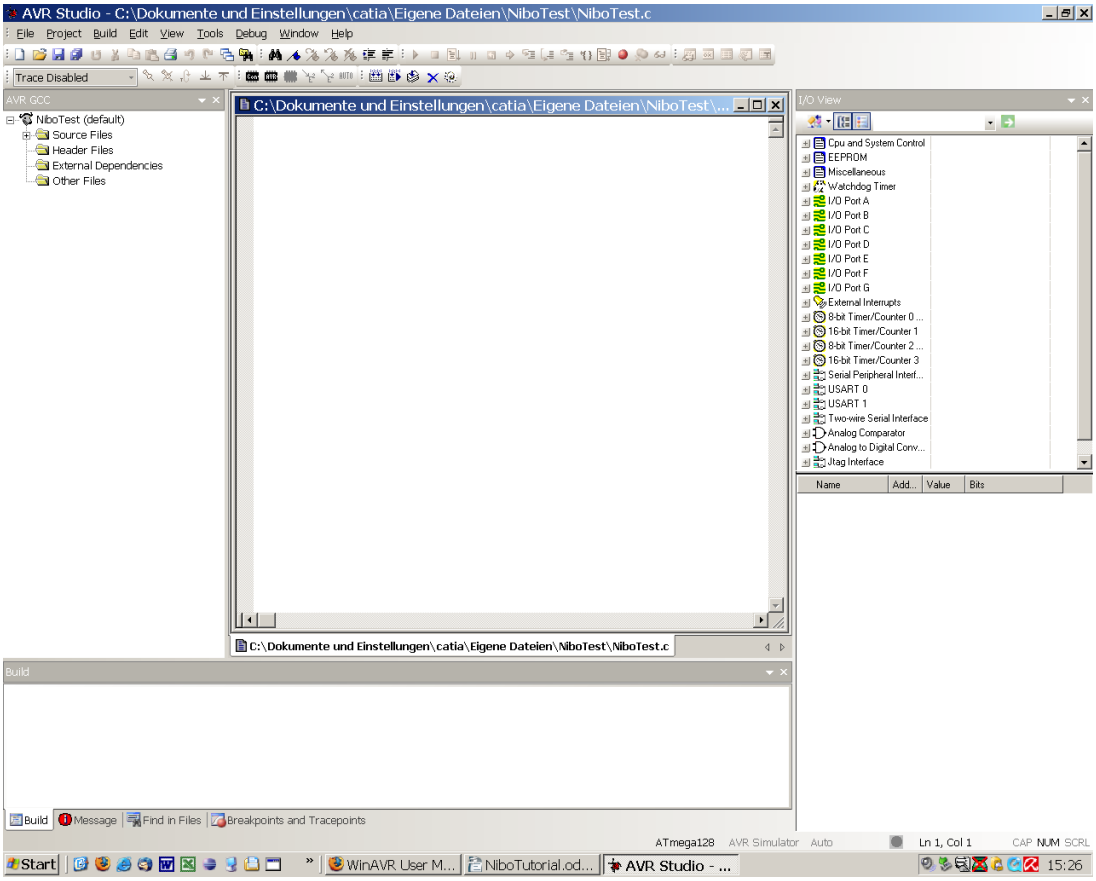


Klicken Sie auf *Next*. Im folgenden Bildschirm wählen Sie als *Debug platform* **AVR Simulator** und als *Device* **ATmega128** aus und klicken auf *Finish*.



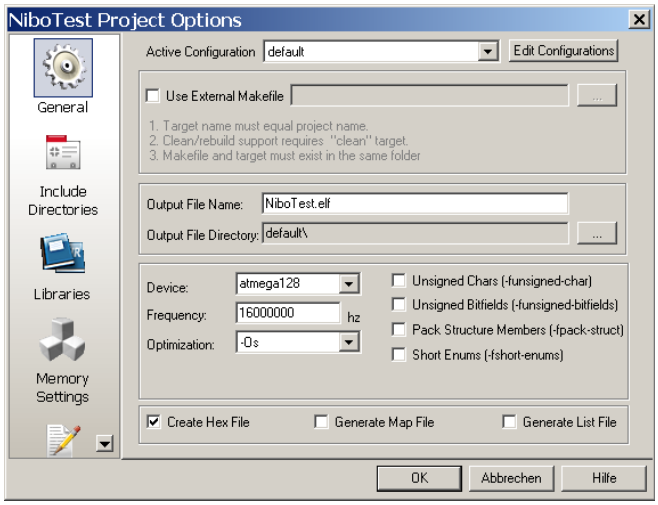
Unser neues Projekt „NiboTest“ ist nun angelegt!

Das AVR Studio sollte in etwa folgender Abbildung gleichen:



Nun müssen noch einige Voreinstellungen eingegeben werden.

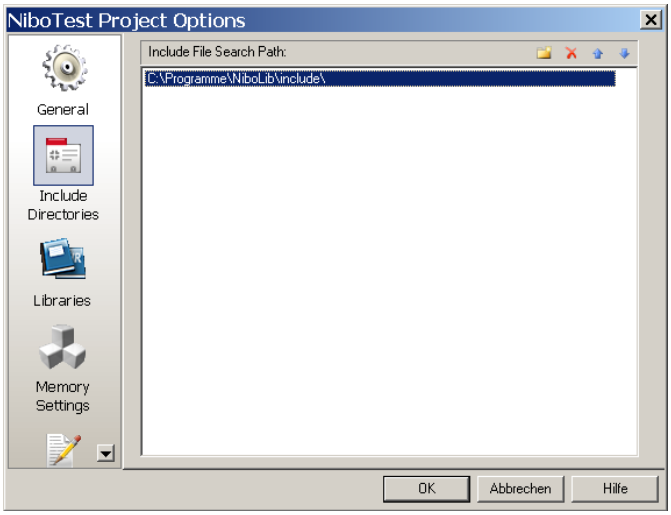
Öffnen Sie dazu den *Project Options* Dialog: In der Menüleiste auf *Project -> Configuration Options* klicken.



Tragen Sie im Bereich *General* im Feld *Frequency* den Wert **16000000** ein und wählen Sie im Feld ***O**ptimization* den Wert **-Os** aus.

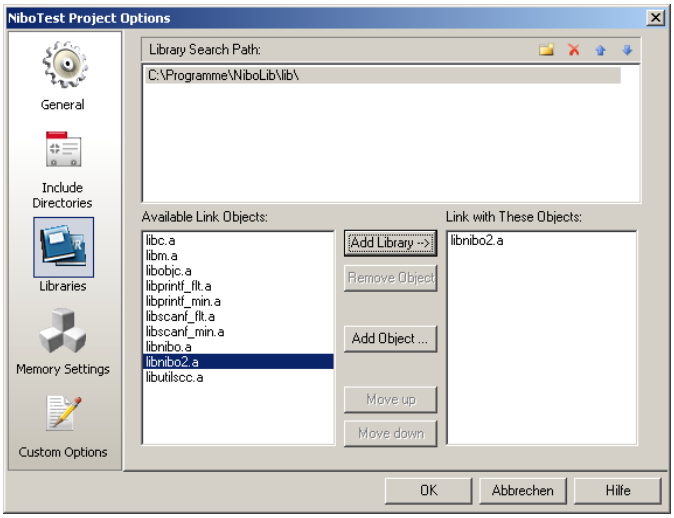
Wählen Sie nun im linken Bereich *Include Directories* aus. Klicken Sie auf das *Neuer Ordner* Symbol und tragen als *Include File Search Path* **C:\Programme\NiboLib\include** ein.

Info: Bei 64-bit Systemen tritt folgender Pfad auf: C:\Programme(x86)\NiboLib\include\

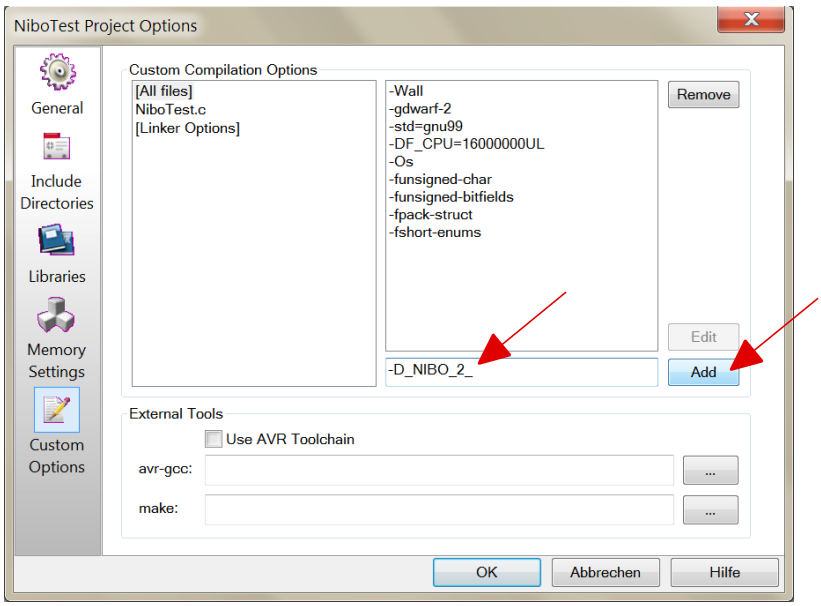


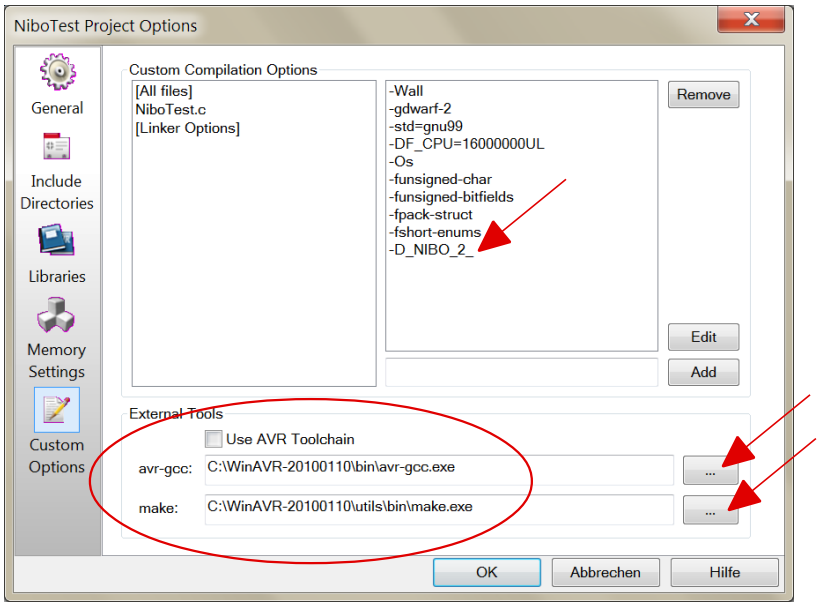
Im Bereich *Libraries* tragen Sie bei *Library Search Path* den Pfad **C:\Programme\NiboLib\lib** ein.

Nun wählen Sie aus den *Available Link Objects* **libnibo2.a** aus und klicken auf den Button *Add Library->*. Die Bibliothek **libnibo2.a** sollte nun im rechten Fenster erscheinen. Bestätigen Sie diese Einstellungen mit OK.



Im Bereich *Custom Options* wird jetzt noch die Option **-D_NIBO_2_** durch Eintragung in das Textfeld und anschließendes Klicken auf *Add* hinzugefügt.





Zusätzlich müssen noch die Pfade für den Compiler und für das Make-File ausgewählt werden (dies geht nur mittels der ***Browse-Buttons***), die Option *Use AVR Toolchain* darf **NICHT** angehakt sein:

avr-gcc: **C:\WinAVR-20100110\bin\avr-gcc.exe**
make: **C:\WinAVR-20100110\utils\bin\make.exe**

Bestätigen Sie die Einstellungen mit OK.

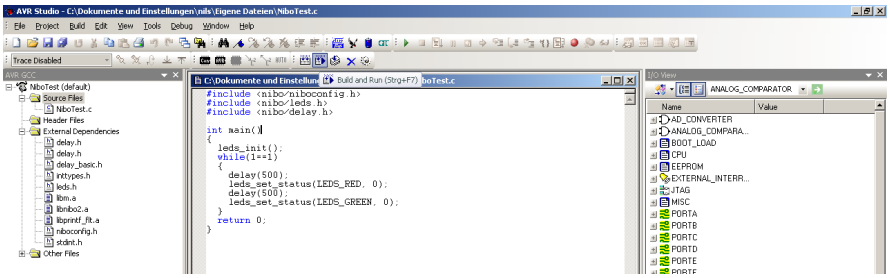
4 Ein erstes Testprogramm

Tippen Sie als erstes Testprogramm folgendes ein:

```
#include <nibo/niboconfig.h>
#include <nibo/leds.h>
#include <nibo/delay.h>

int main() {
  leds_init();
  while(1==1) {
    delay(500);
    leds_set_status(LEDS_RED, 0);
    delay(500);
    leds_set_status(LEDS_GREEN, 0);
  }
  return 0;
}
```

Jetzt wird das Programm compiliert und anschließend im Simulator gestartet, indem Sie den *Build & Run (Strg+F7)* Knopf drücken:

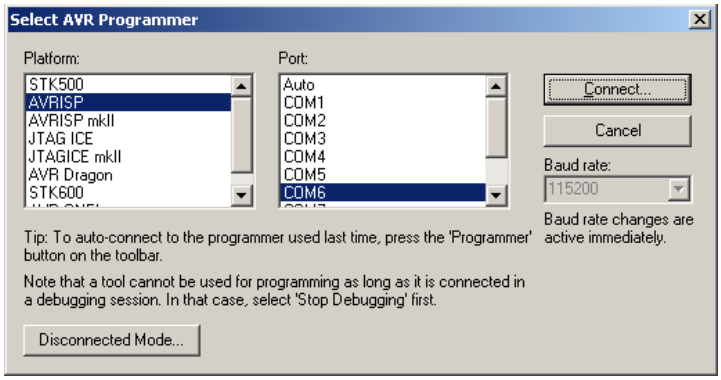


Falls es beim compilieren keinen Fehler gegeben hat können Sie das Programm jetzt vom Simulator auf den Nibo übertragen.

Dafür bietet sich der speziell für den Nibo entwickelte Programmieradapter UCOM-IR an (siehe http://ucom-ir.nicai-systems.de). Alternativ kann auch das STK500 oder der AVR-ISP Programmieradapter von Atmel verwendet werden.

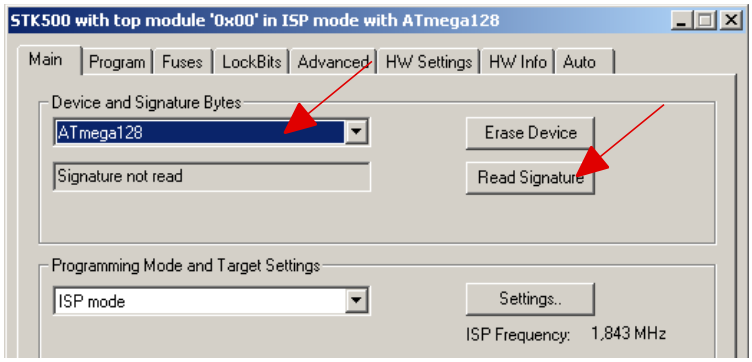
Tip: Da der NIBO2 während des Programmiervorgangs eingeschaltet sein muss, empfiehlt es sich, die Motoren durch Abziehen des Motorjumpers zu deaktivieren!

Um das Programm zu übertragen drücken Sie im AVR Studio auf den *Connect* Button. Anschließend erhalten Sie folgenden Dialog:

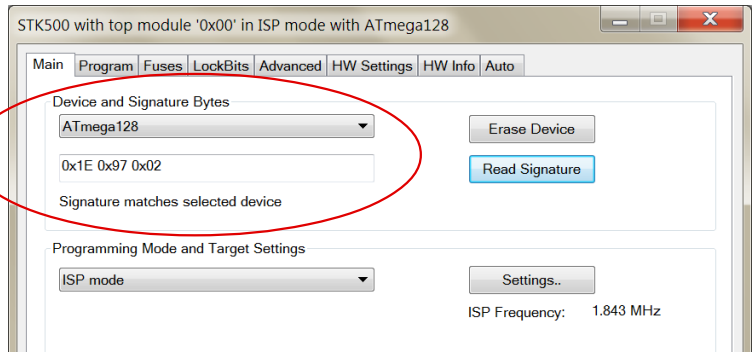


Wählen Sie als *Platform* AVRISP (oder STK500) aus und als *Port* den von Windows zugewiesenen COM-Port aus. Drücken Sie danach *Connect...*

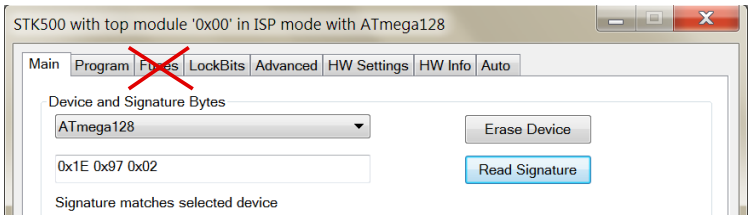
Wählen Sie nun im angezeigtem Dialog „*ATmega128*“ aus und testen Sie die Verbindung indem Sie Read Signature drücken:



Wenn alles geklappt hat, erscheint der Text „*Signature matches selected device*“. Als Device Signature erscheint: „0x1E 0x97 0x02“:

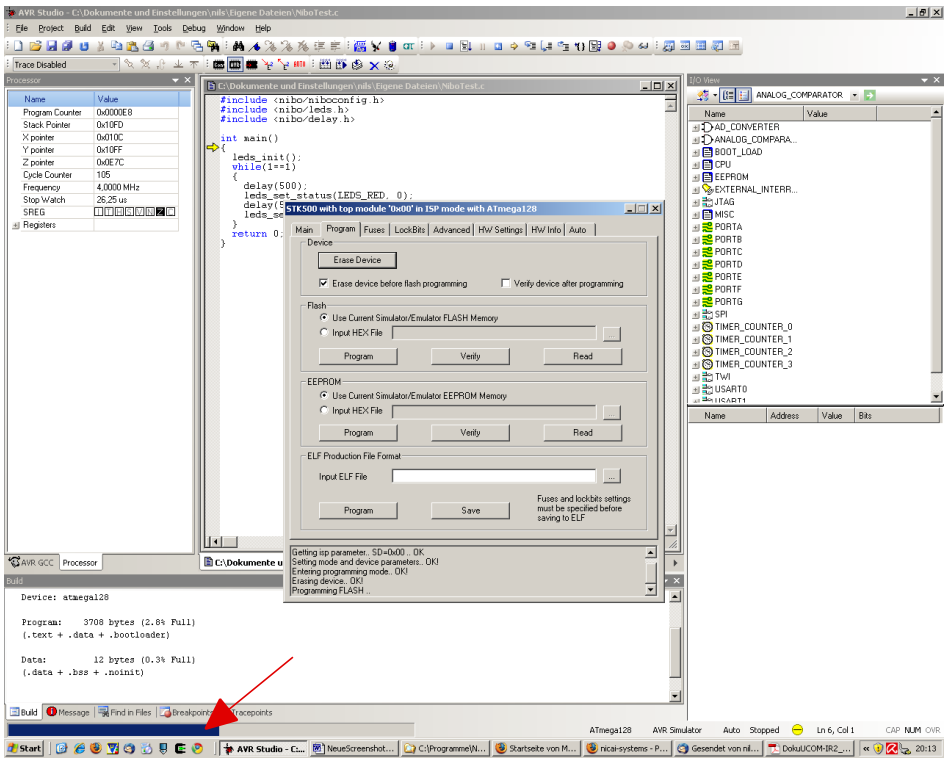


!! Wichtig !!



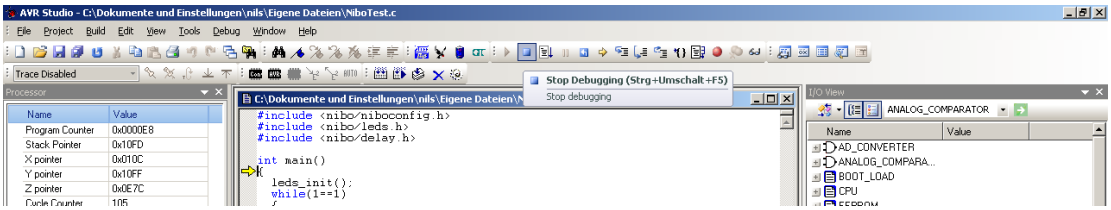
Im Tabsheet „*Fuses*“ dürfen **keine Änderungen** vorgenommen werden! Die Fuse-Bits des ATmega128 sind standardmäßig auf folgende Werte eingestellt: EXTENDED=0xFF, HIGH=0xC1, LOW=0xFF. Diese Werte **dürfen nicht** geändert werden, da man sich sonst aus dem Controller aussperrt!!! Weitere Infos unter: http://www.nibo-roboter.de/wiki/Ausgesperrt

Wechseln Sie nun zum Tabsheet *Program* und haken Sie in der Sektion *Device* die Option ***Erase device before flash programming*** an. Wählen Sie zusätzlich in der Sektion *Flash* die Option ***Use Current Simulator/Emulator FLASH Memory*** aus und klicken Sie in dieser Sektion auf *Program.*



Die Programmübertragung wird mittels Verlaufsbalken am unteren Fensterrand visualisiert.

Der Simulator wird verlassen, indem man in der Toolbar den Button *Stop Debugging (Strg+Umschalt+F5)* anklickt:



Wenn die Übertragung des Testprogramms geklappt hat, sollte am Nibo2 das rechte Rücklicht LED0 abwechselnd rot und grün aufleuchten.

Erläuterungen zum Quellcode:

Die ersten drei Programmzeilen beginnen jeweils mit einer # und sind somit Präprozessoranweisungen. Der Präpozessor wird angewiesen, die in den spitzen Klammern angegebenen header-Dateien (.h-Dateien) einzubinden. Das Schlüsselwort hierfür heißt *include*.

Mit den Zeilen `int main()` und `{` beginnt das Hauptprgramm. Es endet mit *`}`*. Zwischen diesen geschweiften Klammern stehen alle Anweisungen, die abgearbeitet werden sollen.

Mit `leds_init();` werden die IO-Ports für die LEDs initialisiert.

Der letzte Programmteil besteht aus einer while-Schleife `while(1==1)`*`{...}`*. Solange die Bedingung in den runden Klammern, hier `1==1`, wahr ist, wird die while-Schleife ausgeführt. In unserem Beispiel handelt es sich also um eine Endlosschleife.

In den geschweiften Klammern steht der so genannte Anweisungsblock der von der while-Schleife immer wieder ausgeführt wird.

Die Anweisung `delay(500);` weist den Controller an, 500 ms zu warten.

Die zweite Anweisung `leds_set_status(LEDS_RED, 0);` setzt den Status der LED 0 auf *RED*. Damit leuchtet die LED Nummer 0 (rechtes Rücklicht) rot.

Dann nochmals 500 ms warten.

Die vierte Anweisung `leds_set_status(LEDS_GREEN, 0);` setzt den Status der LED 0 auf *GREEN*. Damit leuchtet die LED Nummer 0 jetzt grün.

Die letzte Anweisung `return 0;` wird nie ausgeführt, da die Endlosschleife nicht verlassen wird. Sie muss jedoch aus formalen Gründen vorhanden sein, da es sonst eine Compiler-Warnung gibt.

Das war´s schon!

5 LEDs in Aktion

Hier wollen wir nun etwas kompliziertere Leuchtmuster gestalten.

Als erstes legen Sie ein neues Projekt mit dem Namen **Leuchtdioden1**, wie in Kapitel 3 beschrieben, an. Denken Sie dabei an die zusätzlichen Einstellungen bei den *Project-Options*.

Die Leuchtdioden sollen nacheinander reihum in einer Farbe eingeschaltet werden. Die Farbe soll bei jedem Umlauf zwischen *Aus*, *Rot*, *Grün* und *Orange* wechseln.

Tippen Sie folgenden Quellcode in das Editorfenster des AVR Studio ein:

```
#include <nibo/niboconfig.h>
#include <nibo/leds.h>
#include <nibo/delay.h>

int main() {
  leds_init();
  while(1==1) {
    int farbe;
    for (farbe=0; farbe<4; farbe++) {
      int ledNr;
      for (ledNr=0; ledNr<8; ledNr++) {
        leds_set_status(farbe, ledNr);
        delay(150);
      }
    }
  }
  return 0;
}
```

Erläuterungen zum Quellcode:

Dieser Quellcode unterscheidet sich nur im Anweisungsblock der main-Funktion von dem Programm aus Kapitel 4.

Es beginnt wieder mit der Initialisierung der LEDs und einer while-Schleife, die „endlos“ läuft.

Mit der ersten Anweisung innerhalb der while-Schleife `int farbe;` wird eine Variable namens „farbe“ vom Typ *int* (Ganzzahl) deklariert.

Als nächstes werden zwei ineinander geschachtelte for-Schleifen verwendet.

Die äußere Schleife läuft über den Wert der Variablen `farbe`. Die Variable wird bei 0 initialisiert und nimmt bis zur Abbruchbedingung nacheinander die Werte 0, 1, 2 und 3 an.

Mit der Anweisung `int led;` wird eine Variable namens „ledNr“ vom Typ *int* deklariert.

Diese wird in der inneren Schleife verwendet. In der inneren for-Schleife stehen nun auch die eigentlichen Anweisungen:

Mit `leds_set_status(farbe, ledNr);` wird der LED mit der Nummer *ledNr* die Farbe *farbe* zugewiesen.

Anschließend wird 150 ms gewartet.

Aufgaben/Anregungen:

1. Verändern Sie Ihr Programm so, dass die Leuchtdioden langsamer hintereinander aufleuchten.
2. Verändern Sie Ihr Programm so, dass die Leuchtdioden schneller hintereinander aufleuchten.
3. Verändern Sie Ihr Programm so, dass sich die Laufrichtung umkehrt, sprich die Diode LED0 soll in jedem Durchlauf als letzte Diode aufleuchten.
4. Verändern Sie Ihr Programm so, dass jede Diode einzeln alle Farben durchläuft (Aus, Rot, Grün und Orange), bevor die nächste beginnt.

6 Inbetriebnahme des Displays

Als nächstes wollen wir uns die Ansteuerung des Grafikdisplays vornehmen: Dazu legen Sie wieder ein neues Projekt an, diesmal mit dem Namen **Display** (wie in Kapitel 3 beschrieben). Denken Sie auch diesmal wieder an die zusätzlichen Einstellungen bei den *Project-Options*.

Unser Programm soll den Text „hello world!“ auf dem Display ausgeben.

Tippen Sie folgenden Quellcode in das Editorfenster des AVR Studio ein:

```
#include <nibo/niboconfig.h>
#include <nibo/display.h>
#include <nibo/gfx.h>

int main() {
  display_init();
  gfx_init();
  gfx_move(30, 20);
  gfx_set_proportional(0);
  gfx_print_text("hello world!");
  gfx_move(30, 30);
  gfx_set_proportional(1);
  gfx_print_text("hello world!");
  return 0;
}
```

Erläuterungen zum Quellcode:

Zunächst werden zusätzlich zur schon bekannten Header-Datei *niboconfig.h* zwei neue Header-Dateien *display.h* und *gfx.h* eingebunden. Dies geschieht wieder mit *#include*.

In der main-Funktion wird mit dem Funktionsaufruf `display_init()` *zunächst die Schnittstelle zum Display initialisiert* und anschließend mit der Anweisung `gfx_init()` das Grafikdisplay konfiguriert.

Danach wird mittels `gfx_move(30, 20)` der Stift auf eine Position 30 Pixel vom linken Rand und 20 Pixel vom Oberen Rand verschoben.

Die Anweisung `gfx_set_proportional(0)` sorgt dafür, dass die Texte nicht in Proportionalschrift ausgedruckt werden, das heisst alle ausgegebenen Zeichen haben die gleiche Breite.

Nun kann auch schon mit `gfx_print_text("hello world!")` der Text „hello world!“ ausgegeben werden.

Jetzt wandert der Cursor mit `gfx_move(30, 30)` eine Zeile tiefer. Der Text

soll nun in Proportional-Schrift ausgeben werden, dies geschieht durch den Aufruf der Funktion `gfx_set_proportional(1)`.

Die folgende Anweisung gibt wiederum den Text „hello world!“ auf dem Display aus, nur diesmal in der anderen Schriftart!

Aufgaben/Anregungen:

1. Geben Sie die Texte „links oben“, „rechts oben“, „links unten“ und „rechts unten“ an den jeweiligen Stellen auf dem Display aus.
2. Geben Sie mittig auf dem Display einen Countdown aus. Der Countdown soll im Sekundentakt von 20 auf 0 zählen. Benutzen Sie dazu die `delay()` Funktion und achten Sie auf eine saubere Ausgabe!

7 Linien- / Bodensensoren

In diesem Beispiel wollen wir uns mit den beiden Boden- und den beiden Liniensensoren beschäftigen. Die Sensoren arbeiten nach dem IR-Reflexionsverfahren. Die Helligkeit des Bodens wird zweimal gemessen, einmal bei eingeschalteter und einmal mit ausgeschalteter IR-LED. Dadurch lassen sich die Einflüsse des Umgebungslichts minimieren.

Die gemessenen Werte werden von der Bibliothek normalisiert und stehen anschließend im Array `floor_relative` zur Verfügung. Damit die Normalisierung durchgeführt werden kann, müssen die Sensoren zuvor kalibriert werden. Die Parameter der Kalibrierung werden im EEPROM dauerhaft (EESAVE=0) gespeichert.

7.1 Kalibrierung der Linien- / Bodensensoren

Zur Kalibrierung dient das Programm **calibration.hex** aus dem Verzeichnis **C:\Programme\NiboLib\hex**.

Um das Programm zu übertragen drücken Sie im AVR Studio auf den *Connect* Button. Wechseln Sie im anschließenden Dialog zum Tabsheet *Program* und haken Sie in der Sektion *Device* wieder die Option ***Erase device before flash programming*** an.

Wählen Sie zusätzlich in der Sektion *Flash* die Option ***Input HEX File*** und als Datei **C:\Programme\NiboLib\hex\calibration.hex** aus. Klicken Sie zur Übertragung auf den Button *Program* in der selben Sektion.

Nach abgeschlossener Übertragung können Sie die Sensoren kalibrieren:

- *Rechtes Rücklicht leuchtet*: Stellen Sie den Roboter auf einen **schwarzen Untergrund** und drücken Sie den Taster S3.
- *Linkes Rücklicht leuchtet*: Stellen Sie den Roboter auf einen **weißen Untergrund** und drücken Sie den Taster S3.
- *Beide Rücklichter leuchten*: Drücken Sie den Taster S3 um die **Parameter** im EEPROM zu **speichern**.

7.2 Anzeige der Werte der Linien- / Bodensensoren

Legen Sie wie in den vorangegangenen Beispielen ein neues Projekt an. Zu den Einstellungen aus den vorherigen Beispielen müssen Sie dieses Mal in den Projektoptionen neben der Bibliothek **libnibo2.a** zwei zusätzliche Bibliotheken zum Linken auswählen: **libm.a** und **libprintf_flt.a**. Weiterhin

muss bei den Projektoptionen unter *Custom Options -> Linker Options* die Option „**`-Wl,-u,vfprintf`**“ hinzugefügt werden.

Tippen Sie anschließend folgenden Quellcode in das Editorfenster ein:

```
#include <nibo/niboconfig.h>
#include <nibo/display.h>
#include <nibo/gfx.h>
#include <nibo/delay.h>
#include <nibo/iodefs.h>
#include <nibo/bot.h>
#include <nibo/floor.h>
#include <stdio.h>

int main() {
  bot_init();
  floor_init();
  display_init();
  gfx_init();

  gfx_move(22, 0);
  gfx_set_proportional(1);
  gfx_print_text("Floor sensor test");
  gfx_set_proportional(0);
  gfx_move(5, 10);
  gfx_print_char('R');
  gfx_move(118, 10);
  gfx_print_char('L');

  while (1==1) {
    delay(10);
    char text[20]="--  --  --  --  --";

    // Bodensensoren
    floor_update();
    sprintf(text, "%02x  %02x  %02x  %02x",
        (uint16_t)(floor_absolute[FLOOR_RIGHT]/8),
        (uint16_t)(floor_absolute[LINE_RIGHT]/8),
        (uint16_t)(floor_absolute[LINE_LEFT]/8),
        (uint16_t)(floor_absolute[FLOOR_LEFT]/8));
    gfx_move(22, 30);
    gfx_print_text(text);

    sprintf(text, "%02x  %02x  %02x  %02x",
        (uint16_t)(floor_relative[FLOOR_RIGHT]/8),
        (uint16_t)(floor_relative[LINE_RIGHT]/8),
        (uint16_t)(floor_relative[LINE_LEFT]/8),
        (uint16_t)(floor_relative[FLOOR_LEFT]/8));
    gfx_move(22, 40);
    gfx_print_text(text);

    // Spannung
    bot_update();
```

```
    float volt = 0.0166 * bot_supply - 1.19;
    sprintf(text, "%3.1fV", (double)volt);
    gfx_move(30, 10);
    gfx_set_proportional(1);
    gfx_print_text("supply: ");
    gfx_set_proportional(0);
    gfx_print_text(text);
  }

  return 0;
}
```

Erläuterungen zum Quellcode:

Diesmal müssen wir am Anfang einen ganze Menge Header-Dateien einbinden, da in diesem Beispiel auf die meisten Teilsysteme des Roboters zugegriffen wird.

`main`-Funktion:
Um die Messung der Versorgungsspannung zu ermöglichen muss zunächst die Funktion `bot_init()` aufgerufen werden. Die Funktion `floor_init()` initialisiert die Boden- und Liniensensoren, die Funktionen `display_init()` und `gfx_init()` initialisieren das Grafikdisplay. Anschließend werden noch ein paar Texte auf dem Display ausgegeben.

`while`-Schleife:
In der ersten Zeile warten wir erst einmal 10 ms.

In der zweiten Zeile definieren wir uns Speicherplatz für einen 20 Zeichen langen Text, den wir erst einmal mit dem Text „-- -- -- -- --“ initialisieren.

Als nächstes aktualisieren wir die Werte der Boden- und Liniensensoren durch Aufruf der Funktion `measure_ground()`. Die normalisierten Werte liegen zwischen 0 (schwarz) und 1024 (weiss), sie können jedoch auch größer sein, wenn die Kalibrierung auf einem dunkleren weiss durchgeführt wurde. Wir teilen die Werte durch 8, damit sie zwischen 0 und 127 (oder maximal 255) liegen und somit durch 2 hexadezimalen Ziffern dargestellt werden können. Die absoluten Helligkeiten deben wir an der Position x=22 und y=30 aus. Die relativen Werte (unter Berücksichtigung des Umgebungslichtes) geben wir 10 Pixel unter der vorigen Zeile aus.

Als letztes wird durch Aufruf der Funktion `bot_update()` die Versorgungs-spannung gemessen. Der gemessene Wert muss durch die Formel `0,0166 * bot_supply - 1,19` in Volt umgerechnet werden. Die Umwandlung in einen Text geschieht wieder durch Aufruf der Funktion `sprintf()`. Der Text „%3.1f“ bedeutet, dass eine Fließkommazahl mit (mindestens) drei Zeichen ausgegeben werden soll. Dabei soll immer eine Nachkommastelle angezeigt werden. Der fertige Text wird an der Stelle x=25 und y=0 ausgegeben.

8 Distanzsensoren

In diesem Beispiel wollen wir uns mit den Distanzsensoren des Roboters beschäftigen. Die fünf Distanzsensoren werden von dem separaten Mikrocontroller ATmega88 (COPRO) angesteuert. Die Messung erfolgt ähnlich wie bei den Bodensensoren nach dem IR-Reflexionsverfahren. Dabei wird gemessen welcher Anteil vom ausgesendeten Licht zurück reflektiert wird. Alle Messwerte sollen auf dem Grafikdisplay ausgegeben werden.

Zusätzlich soll wieder die aktuelle Versorgungsspannung des Roboters ausgegeben werden.

Legen Sie wie in den vorangegangenen Beispielen ein neues Projekt an. Zu den Einstellungen aus den vorherigen Beispielen müssen Sie dieses Mal in den Projektoptionen neben der Bibliothek **libnibo2.a** zwei zusätzliche Bibliotheken zum Linken auswählen: **libm.a** und **libprintf_flt.a**. Weiterhin muss bei den Projektoptionen unter *Custom Options -> [Linker Options]* die Option „**`-Wl,-u,vfprintf`**“ hinzugefügt werden.

Tippen Sie anschließend folgenden Quellcode in das Editorfenster ein:

```
#include <nibo/niboconfig.h>
#include <nibo/display.h>
#include <nibo/gfx.h>
#include <nibo/copro.h>
#include <nibo/delay.h>
#include <nibo/iodefs.h>
#include <nibo/bot.h>
#include <nibo/spi.h>
#include <avr/interrupt.h>
#include <stdio.h>

int main() {
  sei();
  bot_init();
  spi_init();
  display_init();
  gfx_init();

  gfx_move(15, 0);
  gfx_set_proportional(1);
  gfx_print_text("Distance sensor test");
  gfx_set_proportional(0);
  gfx_move(5, 10);
  gfx_print_char('R');
  gfx_move(118, 10);
  gfx_print_char('L');

  delay(50);
  copro_ir_startMeasure();
```

```
  while (1==1) {
    delay(10);
    char text[20]="--  --  --  --  --";

    // Co-Prozessor
    if (copro_update()) {
      sprintf(text, "%02x  %02x  %02x  %02x  %02x",
          (uint16_t)copro_distance[0]/256,
          (uint16_t)copro_distance[1]/256,
          (uint16_t)copro_distance[2]/256,
          (uint16_t)copro_distance[3]/256,
          (uint16_t)copro_distance[4]/256);
    }
    gfx_move(10, 55);
    gfx_print_text(text);

    // Spannung
    bot_update();
    float volt = 0.0166 * bot_supply - 1.19;
    sprintf(text, "%3.1fV", (double)volt);
    gfx_move(30, 10);
    gfx_set_proportional(1);
    gfx_print_text("supply: ");
    gfx_set_proportional(0);
    gfx_print_text(text);

  }

  return 0;
}
```

Erläuterungen zum Quellcode:

Diesmal müssen wir am Anfang einen ganze Menge Header-Dateien einbinden, da in diesem Beispiel auf die meisten Teilsysteme des Roboters zugegriffen wird.

`main`-Methode:
Zunächst werden die Interrupts durch Aufruf der Funktion `sei()` aus der AVR Bibliothek <avr/interrupt.h> aktiviert. Dies ist notwendig, damit die Kommunikation über den SPI-Bus funktioniert. Zusätzlich müssen für die Kommunikation noch die beiden Funktionen `bot_init()` und `spi_init()` aufgerufen werden. Die Funktionen `display_init()` und `gfx_init()` initialisieren das Grafikdisplay.

Als nächstes werden noch ein paar Texte auf dem Display ausgegeben, kurz gewartet und durch Aufruf der Funktion `copro_ir_startMeasure()` die Distanzmessung gestartet.

`while`-Schleife:
In der ersten Zeile warten wir erst einmal 10 ms.

In der zweiten Zeile definieren wir uns Speicherplatz für einen 20 Zeichen langen Text, den wir erst einmal mit dem Text „-- -- -- -- --“ initialisieren.

Als nächstes aktualisieren wir die Distanzwerte durch Aufruf der Funktion `copro_update()` und benutzen den Rückgabewert in der if-Anweisung, um den folgenden Block nur nach einem erfolgreichen Update auszuführen.

Bei erfolgreichem Update werden die Werte durch Aufruf der Funktion `sprintf()` aus der C-Bibliothek <*stdio.h*> in den Textpuffer geschrieben. Die Zeichenkette dient dabei als Schablone für den Text: „%02x“ bedeutet das 2 hexadezimale Ziffern, im Bedarfsfall mit führender Null, ausgegeben werden.

In jedem Fall gehen wir jetzt zur Position x=10 und y=55 und geben den Text aus. Zum Abschluss wird, wie im vorangegangen Kapitel, die Versorgungs-spannung ausgegeben.

9 Bewegung – Ab die Post!

Im letzten Beispiel wollen wir etwas Bewegung ins Spiel bringen: Der Roboter soll ein Stück vorwärts fahren, kurz warten, dieselbe Strecke rückwärts fahren, wieder kurz warten und von vorne beginnen. Die Motoransteuerung wird auch vom COPRO durchgeführt.

Dabei soll auch diesmal ständig die Batteriespannung auf dem Display ausgegeben werden.

Wie im vorigen Beispiel legen wir wieder ein neues Projekt an und wählen die drei Bibliotheken **libnibo2.a**, **libm.a** und **libprintf_flt.a** aus und fügen bei den Projektoptionen die Option „**`-Wl,-u,vfprintf`**“ hinzu (siehe Kapitel 7)**.**

Schreiben Sie anschließend folgenden Quellcode in das Editorfenster:

```
#include <nibo/niboconfig.h>
#include <nibo/display.h>
#include <nibo/gfx.h>
#include <nibo/copro.h>
#include <nibo/delay.h>
#include <nibo/iodefs.h>
#include <nibo/bot.h>
#include <avr/interrupt.h>
#include <nibo/spi.h>
#include <stdio.h>


int main() {
  sei();
  bot_init();
  spi_init();
  display_init();
  gfx_init();

  gfx_move(62, 0);
  gfx_set_proportional(1);
  gfx_print_text("motion");
  gfx_set_proportional(0);

  gfx_move(5, 0);
  gfx_print_char('R');

  gfx_move(118, 0);
  gfx_print_char('L');

  delay(50);
  copro_ir_startMeasure();
  copro_setSpeedParameters(5, 6, 7);
```

```
  int counter=0;

  while (1==1) {
    delay(10);
    char text[20]="";

    bot_update();
    float volt = 0.0166 * bot_supply - 1.19;
    sprintf(text, "%3.1fV", (double)volt);
    gfx_move(25, 0);
    gfx_print_text(text);

    switch(++counter) {
      case 200:
        copro_setSpeed(20, 20);
        break;

      case 400:
        copro_stop();
        break;

      case 600:
        copro_setSpeed(-20, -20);
        break;

      case 800:
        copro_stop();
        counter=0;
        break;
    }
  }
  return 0;
}
```

Erläuterungen zum Quellcode:

Auch dieses Programm beginnt mit dem Einbinden der verschiedenen Header-Dateien.

Das Hauptprogramm aktiviert zuerst – wie im letzten Beispiel – die Interrupts und initialisiert die verschiedenen Subsysteme. Danach werden einige Texte ausgegeben. Nach einer kurzen Pause die Messung der Distanzwerte gestartet und die Parameter für die Motorregelung an den COPRO übermittelt. Mit den Regelungsparametern `5,6,7` lassen sich gute Ergebnisse erzielen.

Für die `while`-Schleife gibt es diesmal eine Variable `counter`, die die Durchläufe der Endlosschleife zählt.

In der Endlosschleife aktualisieren wir zunächst die Werte vom COPRO.

Im nächsten Abschnitt geben wir, wie im vorangegangenen Beispiel, die aktuelle Versorgungsspannung aus.

Der anschließende `switch`-Block ist die zentrale Steuerung für die Bewegung: Die Variable `counter` wird hochgezählt; bei Erreichen der Werte 200, 400, 600 und 800 werden die entsprechenden Abschnitte ausgeführt.

Von 0-199 sollen sich die Räder nicht drehen.

Von 200-399 sollen sich beide Räder des Roboters mit einer Geschwindigkeit von 20 Ticks pro Sekunde vorwärts drehen.

Von 400-599 sollen sich die Räder nicht drehen.

Von 600-799 sollen sich beide Räder des Roboters mit einer Geschwindigkeit von 20 Ticks pro Sekunde rückwärts drehen.

Bei Erreichen des Wertes 800 wird der Zähler auf 0 zurückgesetzt.

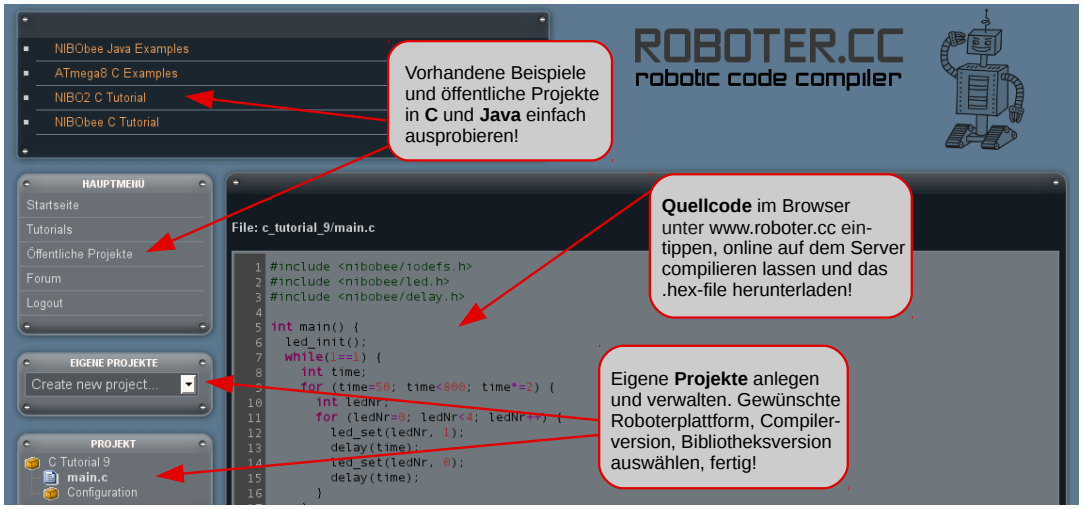
Aufgaben/Anregungen:

1. Lassen Sie den Roboter eine kurze Zeit lang im Uhrzeigersinn drehen, und anschließend gegen den Uhrzeigersinn.
2. Fahren Sie mit dem Nibo ein Quadrat ab. Benutzen Sie hierzu die Funktion `copro_setTargetRel(left, right, speed)`. Mit dieser Funktion ist es möglich, die Räder um eine bestimmte Anzahl von Ticks zu bewegen.

10 Roboter.CC – Robotik Online Code Compiler

Roboter.CC ist eine **alternative Plattform**, auf der eigene Roboter-Projekte **verwaltet** und **compiliert** werden können. Die Installation einer lokalen Entwicklungsumgebung ist nicht notwendig - die Verlinkung der Bibliotheken erfolgt automatisch.

1. Roboter-Typ und gewünschte Programmiersprache auswählen
2. Programmcode schreiben
3. Erzeugte XHEX-Datei mit RoboDude auf den Roboter übertragen



`http://www.roboter.cc`

Roboter-Projekte online compilieren& verwalten, Beispiele einfach ausprobieren

11 Links zu weiterführenden Internetseiten

In diesem Unterkapitel ist eine ausgewählte Linksammlung zu themenähnlichen Internetseiten aufgeführt.

Entwicklungsumgebungen:

- Atmel: http://www.atmel.com Webseite vom Hersteller der Mikrocontroller. Dort gibt es Datenblätter, Applikationsbeispiele und die Entwicklungsumgebung AVRStudio.
- WinAVR: http://winavr.sourceforge.net/ AVR-GCC Compiler für Windows mit vielen Extras und „Add-on“ für das AVRStudio.
- AVRDude: http://savannah.nongnu.org/projects/avrdude/ Freie Programmiersoftware (Downloader, für den Nibo geeignet!).
- Roboter.CC: http://www.roboter.cc Online Code Compiler speziell für Robotik-Projekte mit vielen Beispielen und Forum.

Weitere Informationen:

- Nibo Hauptseite: http://nibo.nicai-systems.de Die Homepage des Nibo Herstellers. Liefert technische Informationen, die Bauanleitung und weitere Links.
- Programmieradapter UCOM-IR: http://ucom-ir.nicai-systems.de
- Nibo Wiki: http://www.nibo-roboter.de Wiki des Nibo. Liefert alle Informationen rund um den Nibo.
- Nibo Shop: http://shop.nicai-systems.de Online-Shop für den Nibo und Erweiterungssets.
- Mikrocontroller: http://www.mikrocontroller.net Alles über Mikrocontroller und deren Programmierung.
- AVRFreaks: http://www.avrfreaks.net Informationen rund um den AVR.