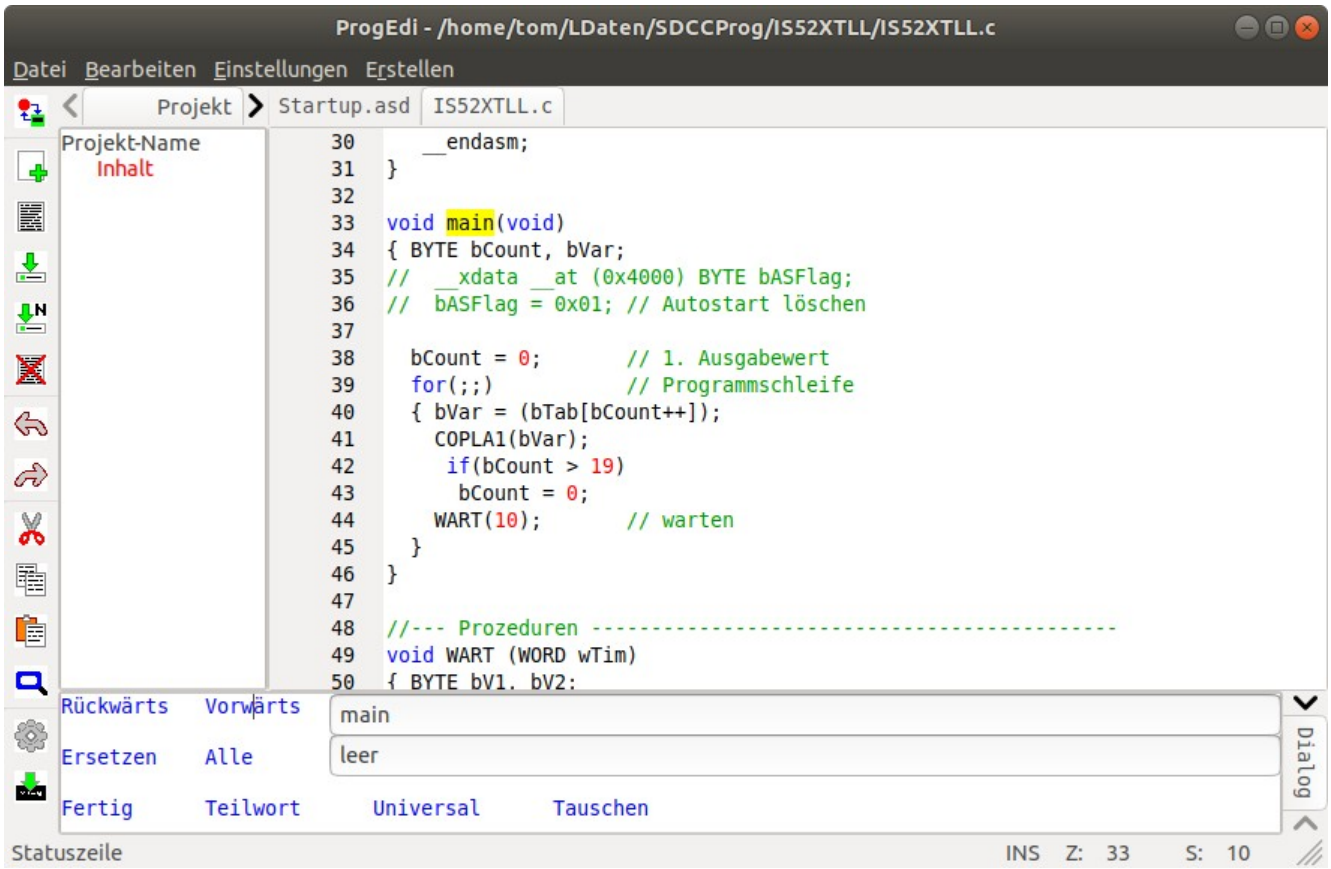
Programmier Editor „ProgEdi“

Bedienungsanleitung

Vorläufig



Stand: 4. Juni 2021
Tom Amann

Inhaltsverzeichnis

1 Einleitung

Ursprünglich war geplant eine Entwicklungsumgebung für Linux, ähnlich Keil µVision(Windows), für bestimmte MCS51-Mikrocontroller zu schreiben. Nach reiflichen Überlegungen entstand aber der Entschluss, die Aufgabe in zwei Teile zu zerlegen. Einen Editorteil, welcher für viele Leute interessant sein kann. Einen Debugger- Simulatorteil, welcher recht speziell und damit nur für mich interessant ist. Dieses Dokument beschreibt den Editorteil.

Der Editor ist in weiten Teilen über eine Konfigurationsdatei einstellbar und damit an viele Aufgaben anpassbar. Er ist geeignet Textdateien zu bearbeiten und anschließend verschiedene wählbare Programme darauf anzuwenden. Seine Spezialität sind aber C, Assembler, und Intel-Hex Dateien. Die Arbeitsweise ist ähnlich CodeBlocks oder Geany nur ist der Editor nicht so umfangreich, um dem Debugger/Simulator mehr Platz in Menü und Arbeitsfenster einzuräumen.

Es können mehrere Dateien gleichzeitig geöffnet sein, wobei immer nur eine den Eingabefokus hat. Bestimmte Funktionen sind an den Dateityp gebunden. So ist z.B: das Sytaxhighlighting bei C, ASM und Intel-Hex unterschiedlich.

Um mehr Textzeilen im Editor darstellen zu können, befindet sich die Werkzeugleiste nicht oberhalb, sondern links vom Text.

Projekte werden noch nicht unterstützt, weshalb das Projektfenster ohne Funktion ist. Die Aufgaben des Projekts kann auch ein Makefile erledigen.

Der Programmier-Editor ProgEdi ist nur für das Betriebssystem Linux verfügbar.

Geschrieben und getestet wurde unter Ubuntu-Linux 18.04

1.1 Installation

Der Programmeditor muss nicht installiert werden, es genügt ihn in ein geeignetes Verzeichnis (mit Schreibzugriff) zu kopieren. Beim ersten Start erzeugt er dort seine Konfigurationsdatei (SDCC-IDE.cfg) mit Default-Werten. Die Konfigurationsdatei ist eine reine Textdatei und kann auch mit einem Editor bearbeitet werden. Dies sollte aber nicht mit dem ProgEdi selbst geschehen, da dadurch evtl. wichtige Werte während sie benötigt werden, geändert werden.

Wird der Editor nicht mehr benötigt, ist er und seine Konfigurationsdatei vom Datenträger zu löschen und das wars. Er nimmt keinerlei Änderungen am Betriebsystem vor.

2 Bedienung

Die Bedienung ist wie bei vielen Editoren realisiert und bedarf nur einer kurzen Beschreibung.

2.1 Menü - Datei

Menü – Datei – Neu

Erzeugt und öffnet eine neue Datei mit den Namen „unbenannt.c“.
Dieser Befehl ist auch in der Werkzeugleiste (Toolbar) erreichbar.

Menü – Datei – Oeffnen

Öffnet eine bereits bestehende Datei.
Dieser Befehl ist auch in der Werkzeugleiste erreichbar.

Menü – Datei – Speichern

Speichert die aktuelle Datei.
Dieser Befehl ist auch in der Werkzeugleiste erreichbar.

Menü – Datei – Speichern unter

Speichert die aktuelle Datei unter einem neuen Namen.
Dieser Befehl ist auch in der Werkzeugleiste erreichbar.

Menü – Datei – Schließen

Schließt die aktuelle Datei, wurde sie geändert wird nachgefragt ob gespeichert wird.
Dieser Befehl ist auch in der Werkzeugleiste erreichbar.

Menü – Datei – Beenden

Beendet den Editor.
Dieser Befehl wird auch durch die Tastenkombination Strg+Q erreicht.

2.2 Menü - Bearbeiten

Menü – Bearbeiten – Rückgängig

Macht den letzten Bearbeitungsschritt rückgängig. Gibt es nichts rückgängig zu machen, ist der Pfeil grau. Dieser Befehl ist auch in der Werkzeugleiste (Toolbar) erreichbar.

Menü – Bearbeiten – Wiederholen

Macht den letzten Rückschritt rückgängig. Gibt es nichts rückgängig zu machen, ist der Pfeil grau. Dieser Befehl ist auch in der Werkzeugleiste (Toolbar) erreichbar.

Menü – Bearbeiten – Ausschneiden

Schneidet den markierten Textbereich aus und legt ihn in die Zwischenablage.
Dieser Befehl wird auch durch die Tastenkombination Strg+x erreicht.

 Dieser Befehl ist auch über die Werkzeugleiste (Toolbar) erreichbar.

Menü – Bearbeiten – Kopieren

Kopiert den markierten Textbereich in die Zwischenablage.
Dieser Befehl wird auch durch die Tastenkombination Strg+c erreicht.

 Dieser Befehl ist auch über die Werkzeugleiste (Toolbar) erreichbar.

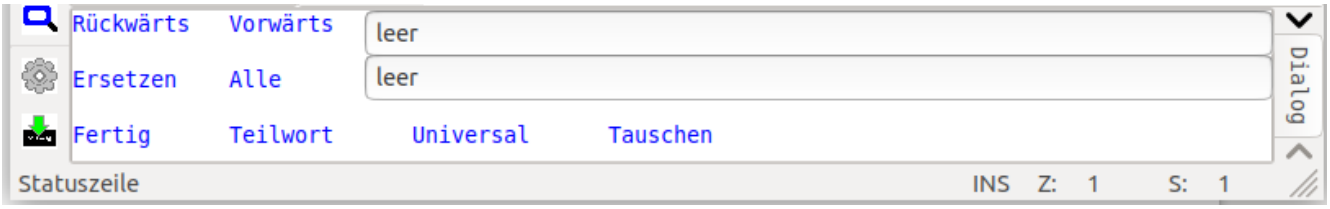
Menü – Bearbeiten – Einfügen

Inhalt der Zwischenablage an der Cursorposition einfügen.
Dieser Befehl wird auch durch die Tastenkombination Strg+v erreicht.

 Dieser Befehl ist auch über die Werkzeugleiste (Toolbar) erreichbar.

Menü – Bearbeiten – Finden

Öffnet den Suchen/Ersetzen Dialog.
Dieser Befehl wird auch durch die Tastenkombination Strg+f erreicht.



Rückwärts: Die Suche erfolgt rückwärts. Ist der Dateianfang erreicht, wird ab dem Ende weitergesucht.

Vorwärts: Die Suche erfolgt vorwärts. Ist das Dateiende erreicht, wird ab dem Dateianfang weitergesucht.

Ersetzen: Das gefundene Wort wird durch das Ersatzwort ersetzt.

Alle: Alle gefundenen Suchwörter werden durch das Ersatzwort ersetzt.

Fertig: Beendet den Suchdialog und zeigt das Statusfenster wieder an. Das Such- und Ersatzwort bleiben bis zum beenden des Programms erhalten und sind beim erneuten Aufruf von „Finden“ wieder präsent.

Teilwort/Vollwort: Entscheidet, ob das Suchwort Teil eines Wortes (Teilwort), oder ein eigenständiges ganzes Wort (Vollwort) ist. Der Modus schaltet bei jedem Klick um (Toggle).

Universal/Sensitiv: Entscheidet, ob zwischen Klein/Großschreibung unterschieden wird (Sensitiv) oder nicht (Universal). Der Modus schaltet bei jedem Klick um (Toggle).

Tauschen: Vertauscht das Such- und Ersatz-Wort. So kann auf die Schnelle ein falsches „alles ersetzen“ Rückgängig gemacht werden. Mit Undo müsste jedes Wort einzeln rückgesetzt werden.

Dieser Befehl ist auch über die Werkzeugleiste (Toolbar) erreichbar.

2.3 Menü - Einstellungen

Die Einstellungen werden in der Konfigurationsdatei gespeichert und bleiben, bis zu einer manuellen Änderung, erhalten. Anstelle von langen Pfadangaben können auch verkürzende Programmvariablen zum Einsatz kommen. Sie beziehen sich immer auf die aktuell aktive Datei:

%c oder %C – Vollständiger Pfad mit Dateinamen und Extend
%p oder %P – Nur der Pfad, ohne Name und Extend
%n oder %N – Nur der Dateiname, ohne Extend
%x oder %X – Nur der Extend

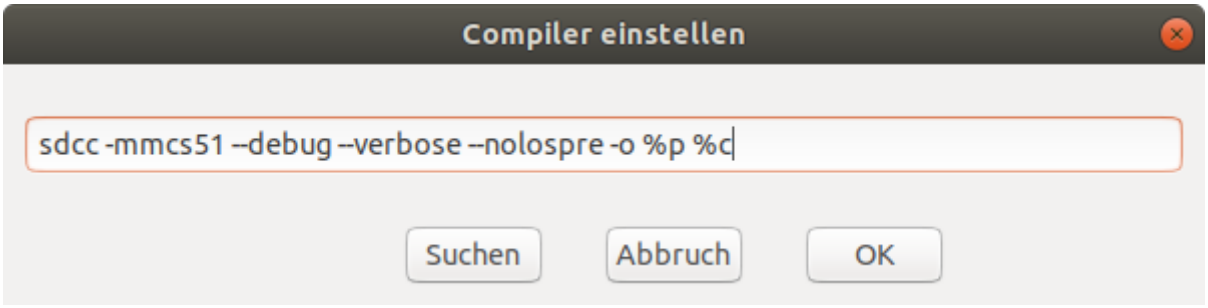
Menü – Einstellungen – Compiler

Hier wird das erste Dienstprogramm mit seinen Aufrufparametern eingestellt. Dieses wird aufgerufen, wenn im Menü – Erstellen – Compilieren geklickt wird. Dies kann ein Compiler oder irgend ein anderes Diensprogramm sein.

Beispiel für SDCC-Compiler für MCS-51:

„sdcc -mmcs51 --debug --verbose --nolospre „

Am Ende der Eingabe muss ein Leerzeichen (Space) stehen, als Trennzeichen für folgende Parameter (z.B: Dateiname).



Bei klick auf „Suchen“ öffnet sich ein „Datei öffnen“ Dialog, um ein Programm mit Pfad auszuwählen. Eventuell benötigte Aufrufparameter sind von Hand anzufügen. Diese Einstellung wird in der Konfig-Datei (Eintrag: C_Com) gespeichert und beim nächsten Programmstart wieder hergestellt.

Menü – Einstellungen – Assembler

Hier wird das zweite Dienstprogramm mit seinen Aufrufparametern eingestellt. Dieses wird aufgerufen, wenn im Menü – Erstellen – Compilieren geklickt wird und eine Assembler-Quelldatei (Extend „.Axx“) aktiv geöffnet ist. Dies kann ein Assembler oder irgend ein anderes Diensprogramm sein.

Beispiel für SDCC-Assembler für MCS-51:

„sdas8051 -plosgff „

Am Ende der Eingabe muss ein Leerzeichen (Space) stehen, als Trennzeichen für folgende Parameter (z.B: Dateiname).

Bei klick auf suchen öffnet sich ein Datei öffnen Dialog, um ein Programm mit Pfad auszuwählen. Eventuell benötigte Aufrufparameter sind von Hand anzufügen. Diese Einstellung wird in der Konfig-Datei (Eintrag: ASM_Com) gespeichert und beim nächsten Programmstart wieder hergestellt.

Menü – Einstellungen – Linker

Hier wird das dritte Dienstprogramm mit seinen Aufrufparametern eingestellt. Dieses wird aufgerufen, wenn im Menü – Erstellen – Linker geklickt wird. Dies kann ein Linker oder irgend ein anderes Diensprogramm sein.

Beispiele für Linker:

„sdcc /home/tom/LDaten/IS52XTLL/IS52XTLL.c /home/tom/LDaten/IS52XTLL/Startup.rel„

oder:
„sdcc %c /home/tom/LDaten/IS52XTLL/Startup.rel“
oder:
„sdcc %c %p/Startup.rel“

Befinden sich die .rel Dateien in einem anderen Verzeichnis, sind die Programmvariablen nicht verwendbar, da sie immer auf den Pfad der gerade aktiven Datei im Editor zeigen. Diese Einstellung wird in der Konfig-Datei (Eintrag: LINK_Com) gespeichert und beim nächsten Programmstart wieder hergestellt.

Menü – Einstellungen – Download

Hier wird das dritte Dienstprogramm mit seinen Aufrufparametern eingestellt. Dieses wird aufgerufen, wenn im Menü – Erstellen – Download geklickt wird. Dies kann ein Downloader oder irgend ein anderes Diensprogramm sein.

Beispiele für SIO-Download:

„/home/tools/./SendFastLX /home/tom/LDaten/IS52XTLL/IS52XTLL.ihx„
oder:
„/home/tools/./SendFastLX %c“
oder:
„/home/tools/./SendFastLX %p/IS52XTLL.ihx“

Hier wird jeweils das Programm „SendFastLX“, aus dem Verzeichnis „/home/tools/“ und als dessen Aufrufparameter die Datei „/home/tom/LDaten/IS52XTLL/IS52XTLL.ihx“, eingestellt.

Bei klick auf „Suchen“ öffnet sich ein Datei öffnen Dialog, um ein Programm mit Pfad auszuwählen. Eventuell benötigte Aufrufparameter sind von Hand anzufügen. Diese Einstellung wird in der Konfig-Datei gespeichert und beim nächsten Programmstart wieder hergestellt. Wenn die Markierung des Eintrags aufgehoben und suchen erneut geklickt wird, kann auch die Datei zum Befehl im Dialog gewählt werden. Diese Einstellung wird in der Konfig-Datei (Eintrag: Down_Com) gespeichert und beim nächsten Programmstart wieder hergestellt.

Menü – Einstellungen – Senden

Hier wird das vierte Dienstprogramm mit seinen Parametern eingestellt. Sonst wie Download. Diese Einstellung wird in der Konfig-Datei (Eintrag: Send_Com) gespeichert und beim nächsten Programmstart wieder hergestellt.

Menü – Einstellungen – Makepfad

Hier wird der Pfad zum Makefile eingestellt. Sonst wie Compiler. Diese Einstellung gilt auch für make clean und wird in der Konfig-Datei (Eintrag: Make_Com) gespeichert und beim nächsten Programmstart wieder hergestellt.

Menü – Einstellungen – Hexmode

Hier wird der Anzeigemodus für Intel-Hex (Extend .HEX oder .IHX) Dateien eingestellt. Der Modus wird bei jedem Klick umgeschaltet. Im Hexmode trägt der Menüitem einen Haken, im Textmode fehlt dieser. Standardmässig ist der Hexmode eingestellt.

Im Textmode wird die Hex-Datei als einfache Textdatei dargestellt. Der Einfügemodus ist Einfügen, kann aber durch die Einfügen-Taste umgeschaltet werden. Anzeige in der Statuszeile „INF“ für einfügen.

Beispiel Textmode:

```
:0341000002410970
:204109007F008F060FEE90416993F582C007124148D007EF24F050027F0090000AC00712C1
:204129004130D00780DC22AE82AF83EE4F600F7D147CC8DCFEDDFA1EBEFF011F80ED22E5AD
:104020000132740132740132740132740132740 14C
:0640FA0075812F02410058
:00000001FF
```

Im Hexmode sind die Elemente der Records farblich unterschiedlich markiert und beim durchsteppen der Daten werden die zwei Zeichen für ein Datenbyte durch einen roten Unterstrich markiert. Bei Änderungen wird die Prüfsumme automatisch mitgeführt. Der Eingabemodus ist auf Überschreiben eingestellt, kann aber durch die Einfügen-Taste umgeschaltet werden. Anzeige in der Statuszeile „OVR“ für überschreiben.

Beispiel Hexmode:

:0341000002410970
:204109007F008F060F<u>EE</u>90416993F582C007124148D007EF24F050027F0090000AC00712C1
:204129004130D00780DC22AE82AF83EE4F600F7D147CC8DCFEDDFA1EBEFF011F80ED22E5AD
:1040200001327401327401327401327401327401 4C
:0640FA0075812F02410058
:00000001FF

Menü – Einstellungen – Baustein

Hier wird die Taktfrequenz des Mikrocontrollers eingestellt. Die Einstellung wird derzeit nicht weiter ausgewertet.

Menü – Einstellungen – MenüInit

Noch nicht vorhanden!

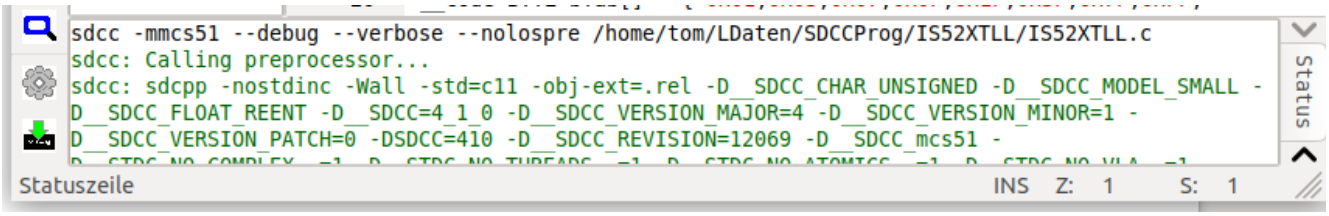
2.4 Menü - Erstellen

Im Menü Erstellen werden die im Menü „Einstellungen“ vorbereiteten Programme aufgerufen.

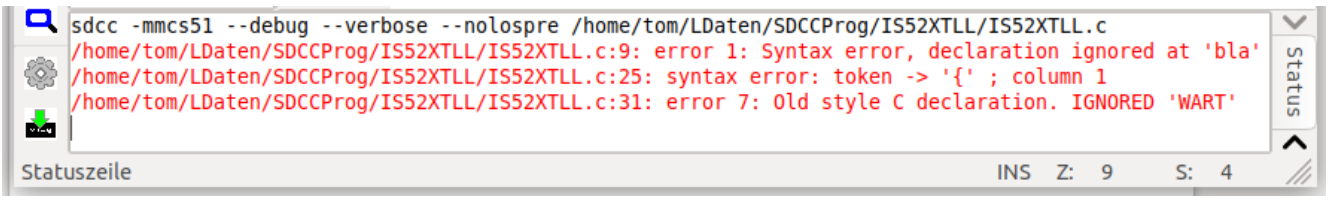
Menü – Erstellen – Compilieren

Hier wird das Programm, welches unter „Menü – Einstellungen – Compiler/Assembler“ eingestellt ist, ausgeführt. Normal-Ausgaben des Programms werden in grüner, Fehlerausgaben in roter, Farbe im Statusfenster angezeigt, sofern das Dienstprogramm dazu in der Lage ist.

Beispiel SDCC – OK:



Beispiel SDCC – Fehler:

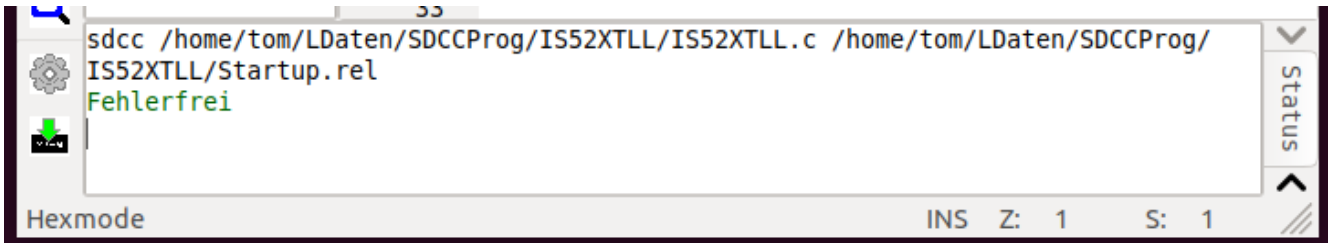


Mit dem Zahnradsymbol kann der Compiler/Assembler aus der Werkzeugleiste gestartet werden. Ob nun der Compiler oder der Assembler gestartet wird, ist abhängig vom Extend der Datei. Ist der erste Buchstabe des Extend ein „C“ wird der Compiler aufgerufen, ist er ein „A“ erfolgt der Aufruf des Assemblers. Hinweis: Der SDCC-Compiler erkennt bei C-Programmen nur den Extend .C nicht .C51.

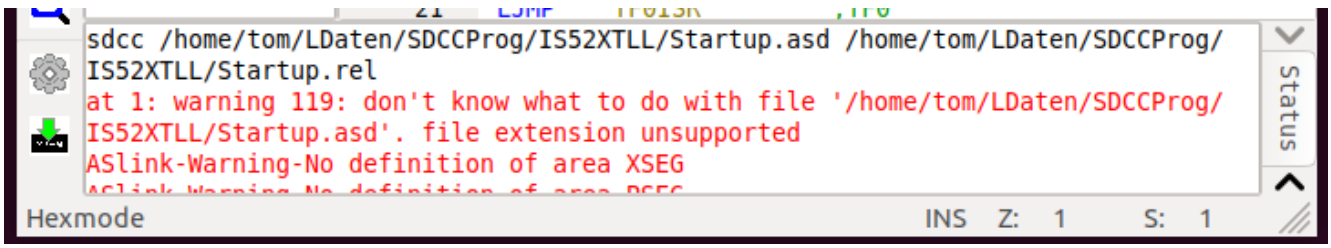
Menü – Erstellen – Linken

Hier wird das Programm, welches unter „Menü – Einstellungen – Linken“ eingestellt ist, ausgeführt. Normal-Ausgaben des Programms werden in grüner, Fehlerausgaben in roter, Farbe im Statusfenster angezeigt, sofern das Dienstprogramm dazu in der Lage ist.

Beispiel Linken – OK:



Beispiel Linken – Fehler:

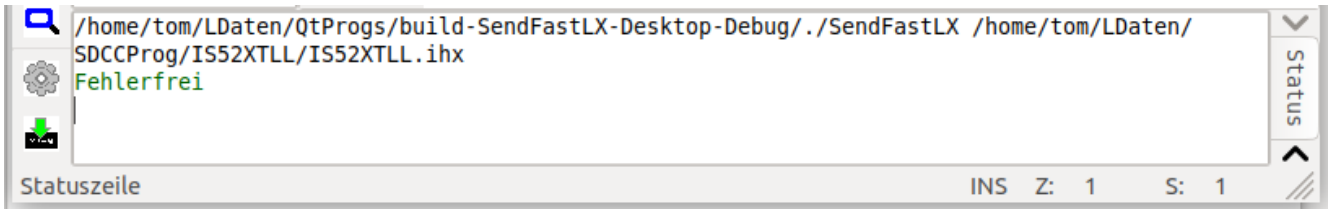


Der Fehler ist, den Linker mit Programmvariablen aus einer aktiven Assemblerdatei heraus, anstelle der C-Datei mit der Main-Funktion, aufzurufen

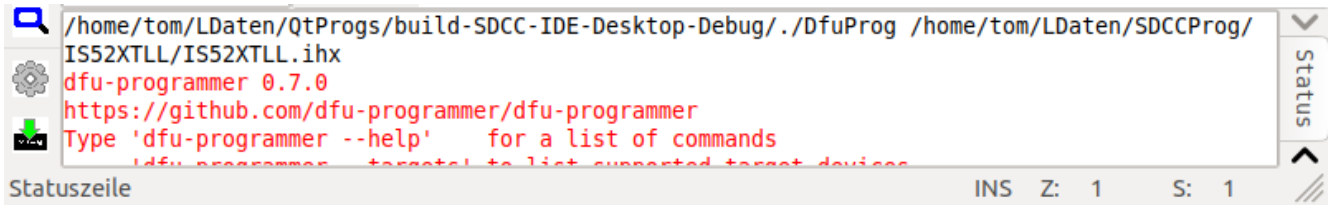
Menü – Erstellen – Download

Hier wird das Programm, welches unter „Menü – Einstellungen – Download“ eingestellt ist, ausgeführt. Normal-Ausgaben des Programms werden in grüner, Fehlerausgaben in roter, Farbe im Statusfenster angezeigt, sofern das Dienstprogramm dazu in der Lage ist.

Beispiel Download – OK:



Beispiel Download – Fehler:



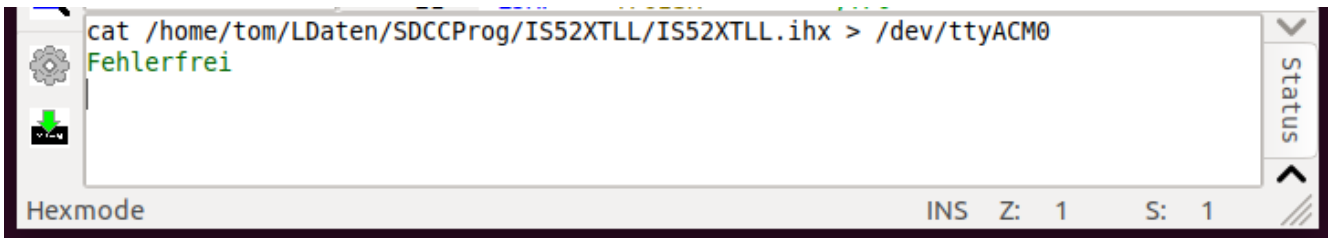
 Mit dem Downloadsymbol kann der Download aus der Werkzeugleiste gestartet werden.

Menü – Erstellen – Senden

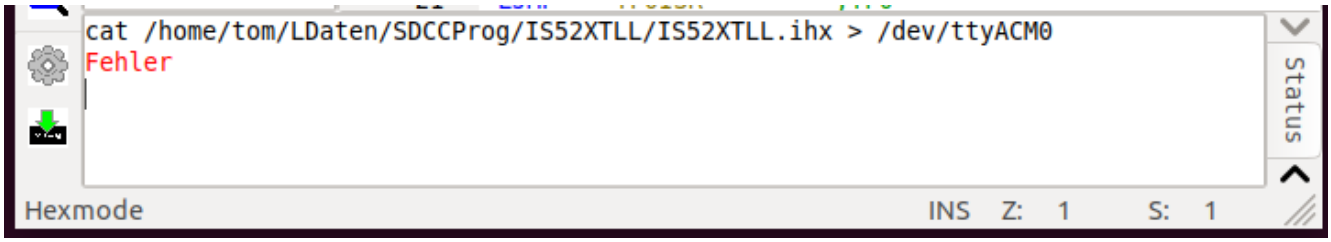
Hier wird das Programm, welches unter „Menü – Einstellungen – Senden“ eingestellt ist, ausgeführt. Normal-Ausgaben des Programms werden in grüner, Fehlerausgaben in roter, Farbe im Statusfenster angezeigt, sofern das Dienstprogramm dazu in der Lage ist. Dies kann z.B: die Übertragung einer Intel-Hex Datei zum Zielsystem sein, wie in Kapitel 5 beschrieben.

Einstellung: **cat %p/IS52XTLL.ihx > /dev/ttyACM0**

Beispiel Senden OK:



Beispiel Senden Fehler:



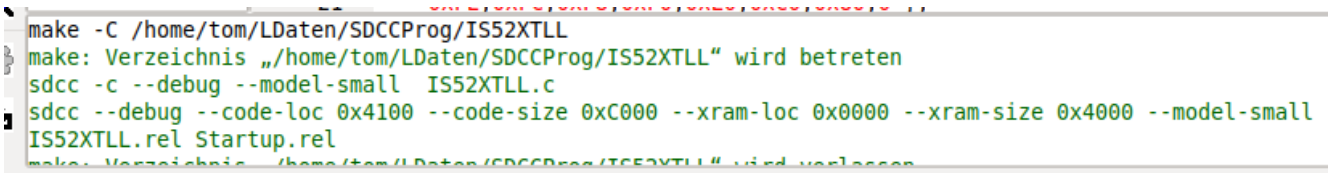
Der Fehler zeigt, dass die Zielhardware nicht angeschlossen ist. Fehler während der Übertragung können hier nicht erkannt werden, da „cat“ einfach nur die Datei sendet und keine Rückmeldung vom Zielsystem erhält.

Menü – Erstellen – xxxxxxx

Hier werden die Programme, welche unter „Menü – Einstellungen – MenüInit“ eingestellt sind, ausgeführt, falls vorhanden. Normal-Ausgaben der Programme werden in grüner, Fehler-ausgaben in roter, Farbe im Statusfenster angezeigt, sofern das Dienstprogramm dazu in der Lage ist.

Menü – Erstellen – Make

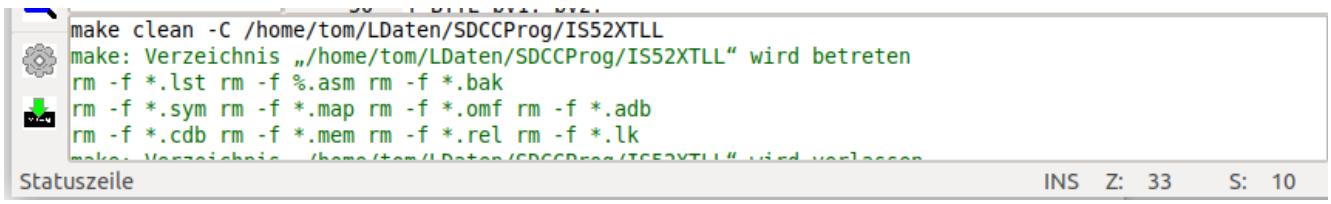
Hier wird das Makefile, welches unter „Menü – Einstellungen – Makepfad“ eingestellt ist, ausgeführt. Normal-Ausgaben des Programms werden in grüner, Fehlerausgaben in roter, Farbe im Statusfenster angezeigt.



Menü – Erstellen – Make Clean

Löscht die im Makefile unter „clean“ aufgeführten Dateien.

$(RM) steht für ReMove (löschen), * steht für alle. Hier werden also alle Dateien mit Extend lst, asm, ……., lk gelöscht. Mit .ASM muss man vorsichtig umgehen, um nicht versehentlich eine eigene Assembler-Quelldatei zu löschen. Aus diesem Grund heißt die Startupdatei „Startup.asd“ (ASD = Assembler Startup Datei). Eine andere Möglichkeit ist, die vom SDC-Compiler erzeugte .asm Datei mit „$(RM) %.asm“ gezielt zu löschen. % steht dabei für den Namen der C-Datei die Main enthält.



3 Sonstiges

Hier werden zusätzliche Funktionen des Programms beschrieben.

3.1 Lesezeichen

Durch Linksklick auf den linken Rand, links des Text und rechts der Zeilennummer (falls vorhanden), wird ein Lesezeichen (Bookmark) in die Zeile eingefügt. Es können auch mehrere Lesezeichen eingefügt sein, aber nur eines je Zeile. Durch nochmaliges anklicken wird das Lesezeichen wieder entfernt, durch erneutes klicken wieder eingefügt (Toggle-Funktion). Sind Lesezeichen gesetzt, lassen sie sich durch die Tastenkombinationen Strg+. (Strg + . Punkt-Taste gleichzeitig) für Vorwärtssuche und Strg+, (Strg + , Komma-Taste gleichzeitig) für Rückwärtssuche anspringen. Die Suche verläuft modulo, ist eine Dateigrenze erreicht, wird am anderen Ende weitergesucht. Damit lassen sich oft gebrauchte Textstellen schnell auffinden und, falls man dort ein Lesezeichen gesetzt hat, schnell zur Ursprungstelle zurückkehren.

Beispiel mit Zeilennummern:

```
13  //--- Funktionsprototypen ----------------------------
14  void WART(WORD wTim);
15  void COPLA1(BYTE bOut);
16  BYTE CIPLA1(void);
17
```

Beispiel ohne Zeilennummern:

```
Sie ist Witwe, denn der Gatte,
den sie hatte, fiel vom Blatte.
Diente so auf diese Weise
einer Ameise als Speise.
```

Die Lesezeichen der geöffneten Dateien werden beim beenden des Programms in der Konfig-Datei gespeichert und beim nächsten Programmstart wieder hergestellt.

3.2 Klammerpaare finden

Durch rechten Mausklick hinter einer Klammer „{[(<>)]}“ wird diese und die dazugehörige Klammer in Gelb auf rotem Hintergrund dargestellt. Befindet sich der Cursor hinter einer Klammer, wird das Paar auch durch die Tastenkombination Strg+k markiert. Die Markierung bleibt erhalten bis ein neues Paar, oder eine Leerstelle markiert wird.

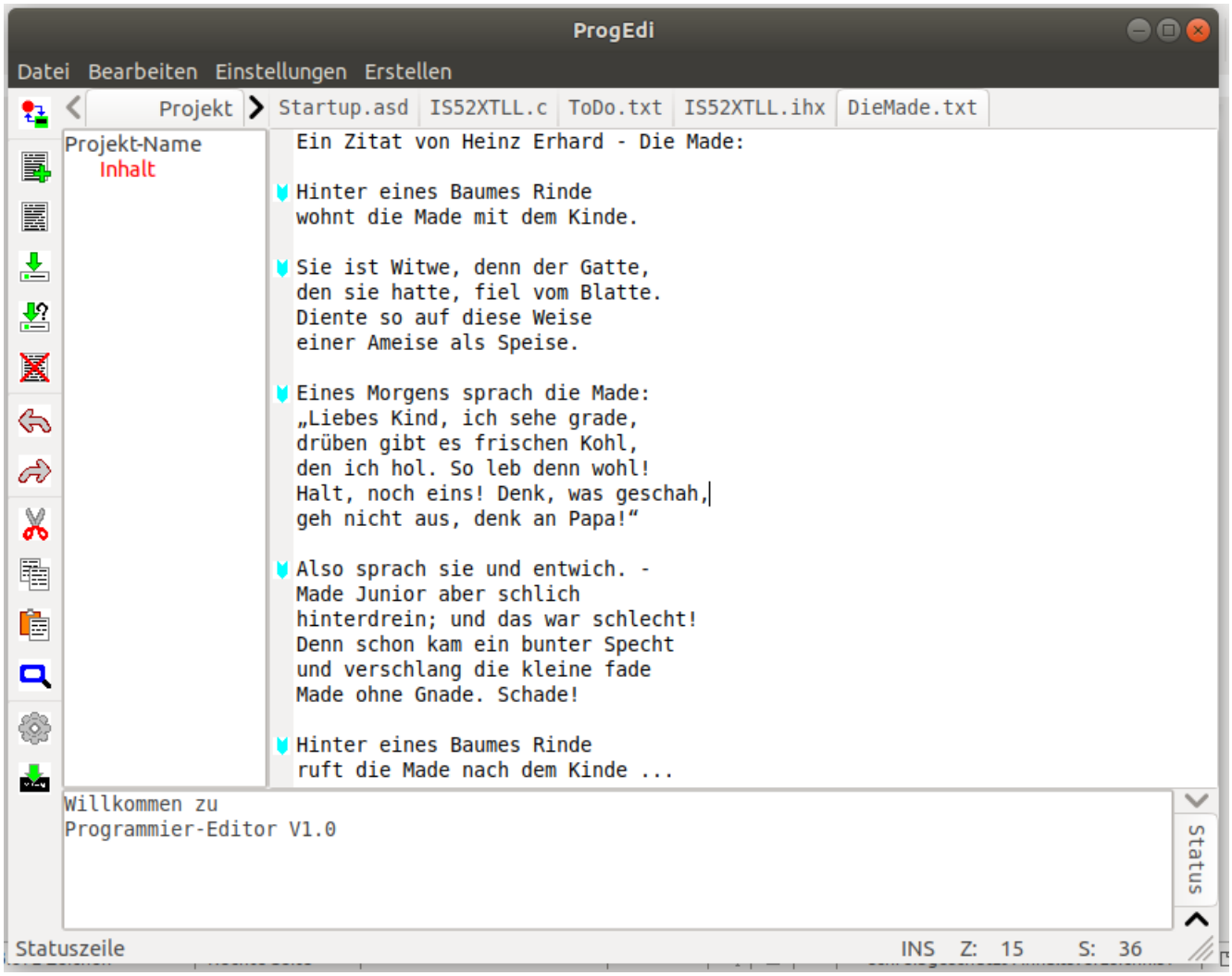
```
52  { for(bV1 = 20; bV1 > 0; bV1--)
53      for(bV2 = 200; bV2 > 0; bV2--);
54  }
```

4 Beispiele

Hier finden sich einige Beispiele für den Umgang mit ProgEdi.

4.1 Einfache Textdatei

Im normalen Textmodus verhält sich ProgEdi wie viele andere Editoren auch. Am linken Rand können Lesezeichen gesetzt sein. Es werden keine Zeilennummern angezeigt. INS steht für INSert (Einfügen), OVR für OVeRwrite (Überschreiben). Die Anzeige der Cursorposition ist unten in der Statuszeile. „Z“ ist die Zeilennummer, „S“ die Spaltennummer der Eingabemarke (Cursor). Im Einfügemodus ist der Cursor ein vertikaler Strich vor der Einfügeposition. Im Überschreibmodus ein horizontaler Strich unter der Einfügeposition.



Der Versuch eine normale Textdatei zu kompilieren/assemblieren endet mit einer Fehlermeldung.

4.2 C-Programmdatei

Hier ein kleines Testprogramm in C für MCS51. Es ließt, in einer Enlosschleife, die Stellung von 8 Schaltern an Port P1 ein und gibt diese an die LED‘s an Port P3 aus.

Programm Prog1.c:

```
/*****************************
 * Testprogramm für C-Compiler *
 *****************************/
#include <8051.h>

int main ()
{ char cVar;
  for(;;)        //Endlosschleife
  { cVar = P1;   //Schalterstellung einlesen
    P3 = cVar;   //Zu den LED's ausgeben
  }
}

Compilieren:     sdcc -mmcs51 --verbose --nolospre -c -o %p/ %c
```

Ausschnitt erzeugte Datei Prog1.lst:

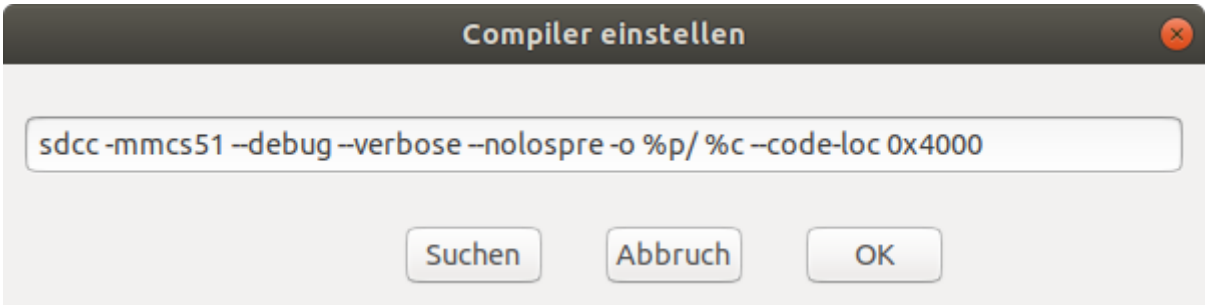
```
000000                   422 00102$:
000000                   423    C$Prog1.c$9$3_0$3 ==.
                         424 ;  { cVar = P1;                 //Schalterstellung einlesen
000000 85 90 B0     [24] 425    mov    _P3,_P1
00003                    426    C$Prog1.c$10$3_0$3 ==.
                         427 ;   P3 = cVar;                 //Zu den LED's ausgeben
000003 80 FB        [24] 428    sjmp   00102$
000005                   429    C$Prog1.c$12$2_0$2 ==.
                         430 ;  }
000005                   431    C$Prog1.c$12$2_0$2 ==.
000005                   432    XG$main$0$0 ==.
000005 22           [24] 433    ret
```

Erzeugte Datei Prog1.ihx:

```
:03000000020006F5
:03005F0002000399
:0300030002006296
:060062008590B080FB2236
:06003500E478FFF6D8FD9F
:200013007900E94400601B7A0090006C780175A000E493F2A308B8000205A0D9F4DAF27527
:02003300A0FF2C
:20003B007800E84400600A790175A000E4F309D8FC7800E84400600C7900900001E4F0A3C3
:04005B00D8FCD9FAFA
:0D0006007581071200 68E5826003020003A7
:04006800758200227B
:00000001FF
```

Diese Hex-Datei ist größer als für den Programmcode benötigt, denn sie beinhaltet auch das initialisieren des Datenspeicher für den Mikrocontroller. Sie kann nun in den Programm-speicher übertragen (Download) und dort ausgeführt werden.

Muss das Programm für eine andere Ursprungsadresse erzeugt werden, so kann diese im Linker festgelegt werden durch --code-loc adresse (z.B: --code-loc 0x4000)
wird die Ursprungsadresse auf 4000hex eingestellt.




```
:03400000002400675
:03405F0002400319
:0340030002406216
:064062008590B080FB22F6
:06403500E478FFF6D8FD5F
:204013007900E94400601B7A0090406C780175A000E493F2A308B8000205A0D9F4DAF275A7
:02403300A0FFEC
:20403B007800E84400600A790175A000E4F309D8FC7800E84400600C7900900001E4F0A383
:04405B00D8FCD9FABA
:0D4006007581071240 68E5826003024003E7
:04406800758200223B
:00000001FF
```

4.2.1 Datenspeicher nicht initialisieren

Beim initialisieren werden Speicherbereiche, und damit Variablen welche sich in diesem Bereich befinden, gezielt mit dem Wert 0 beschrieben. Die im folgenden Listing gezeigte Funktion „nocrtinit“ unterdrückt das initialisieren der Datenspeicher des µC:

```
/*****************************
 * Testprogramm für C-Compiler *
 *****************************/
#include <8051.h>

//--- Initialisierung der Speicher unterdrücken -------
void nocrtinit (void) __naked
{  __asm
   __mcs51_genXINIT::
   __mcs51_genXRAMCLEAR::
   __mcs51_genRAMCLEAR::
   __sdcc_gsinit_startup::
   __endasm;
}

int main ()
{ char cVar;
  for(;;)        //Endlosschleife
  { cVar = P1;   //Schalterstellung einlesen
     P3 = cVar;  //Zu den LED's ausgeben
  }
}
```

Resultierende Intel-Hex Datei:

```
:0340000002400972
:0340060002400372
:034003000240096F
:064009008590B080FB224F
:00000001FF
```

Das erzeugte Programm ist deutlich kleiner, trotz gleichem Quellcode. Zu beachten ist jetzt allerdings, dass der Datenspeicher und damit einige Variablen undefiniert sind.

4.2.2 Mehrere C-Quelldateien

Die meisten Programme bestehen nicht aus einer großen Quelldatei, sondern sind in mehrere Quelldateien aufgeteilt. Das erhöht die Wartbarkeit und Übersicht, aber nur solange man das Programm nicht in zu viele Teilprogramme zersplittert. Hier ein Beispiel für ein Programm das aus zwei Dateien besteht. Die Datei „Prog1Proz.c“ welche Prozeduren (Unterprogramme) für das Hauptprogramm „Prog1.c“ enthält. Die Funktion „main“ befindet sich im Hauptprogramm.

Datei Prog1Proz.c:

```
/*************************
 * Prozeduren für Prog1.c *
 *************************/

void Timer(void)
{ int iZlr = 10000;
  for(; iZlr; iZlr--);
  return;
}
```

Übersetzen durch: sdcc -mmcs51 --debug --verbose --nolospre -c -o %p/ %c
erzeugt die benötigte Datei „Prog1Proz.rel“ für das linken.

Datei Prog1.c:

```
/*****************************
 * Testprogramm für C-Compiler *
 *****************************/
#include <8051.h>

//--- Externe Funktionen -----------------------------
extern void Timer(void);

int main ()
{ char cVar;
  for(;;)        //Endlosschleife
  { cVar = P1;   //Schalterstellung einlesen
      P3 = cVar; //Zu den LED's ausgeben
      Timer();   //Externe Prozedur aufrufen
  }
}
```

Übersetzen durch: sdcc -mmcs51 --debug --verbose --nolospre -c -o %p/ %c
erzeugt die benötigte Datei „Prog1.rel“ für das linken.

Linken der Teilprogramme durch: sdcc -o %p/ %p/%n.rel %p/Prog1Proz.rel
Die Datei, welche die Funktion „main“ enthält, muss als Erste aufgeführt sein. Auf die gleiche Art können mehrere Dateien übersetzt und gelinkt werden. Näheres dazu findet sich im „SDCC User Guide“.

4.3 ASM-Programmdatei

Hier ein kleines Testprogramm für MCS51 in SDCC-Assembler (SDAS8051). Es ließt, in einer Enlosschleife, die Stellung von 8 Schaltern an Port P1 ein und gibt diese an die LED‘s an Port P3 aus. Die Anweisung „.area HOME (ABS)“ weist das Programm dem Codespeicher zu. Die Anweisung „.org 0“ veranlasst die Übersetzung für Ursprungsadresse 0000h. Näheres ist in der Dokumentation des Assemblers (Asxxxx.txt) zu finden. Der Assembler wird automatisch zusammen mit dem Compiler installiert.

Prog1.asm

```
;*****************************
;* Testprogramm für Assembler *
;*****************************
.area HOME (ABS)
.org 0

Start:
 mov a,P1        ;Schalterstellung einlesen
 mov P3,a        ;Zu den LED's ausgeben
 sjmp Start      ;Endlos wiederholen
```

Übersetzt wird das Programm im Editor durch: sdas8051 -plosgff

Der Assembler erzeugt, unter Anderen, die Datei „Prog1.lst“, welche auch die OP-Codes und Speicheradressen enthält:

```
                                 1 ;*****************************
                                 2 ;* Testprogramm für Assembler *
                                 3 ;*****************************
                                 4 .area HOME (ABS)
000000                           5 .org 0
                                 6
000000                           7 Start:
000000 E5 90            [12]     8  mov a,P1        ;Schalterstellung einlesen
000002 F5 B0            [12]     9  mov P3,a        ;Zu den LED's ausgeben
000004 80 FA            [24]    10  sjmp Start      ;Endlos wiederholen
```

Der Linker erzeugt aus der .rel Datei die Intel-Hex Datei. Dem Linker wird die .rel Datei mit ihrem Pfad (%p/%n.rel) übergeben und mit „-o %p/“ werden die erzeugten Dateien im verwendeten Arbeitsverzeichnis abgelegt.

Aufruf: sdcc %p/%n.rel -o %p/ bedeuted: (sdcc Pfad/Dateiname.rel -o Pfad/)

Dadurch wird auch die zum Download benötigte Intel-Hex Datei erzeugt.

Prog1.ihx: :06000000E590F5B080FA66
:00000001FF

Muss das Programm für eine andere Ursprungsadresse erzeugt werden, so kann diese durch die „.org“ Anweisung gewählt werden. Im Beispiel wird das Programm für Ursprungsadresse 4000hex übersetzt.

Beispielprogramm Prog1.asm:

```
;*****************************
;* Testprogramm für Assembler *
;*****************************
.area HOME (ABS)
.org 0x4000

Start:
 mov a,P1        ;Schalterstellung einlesen
 mov P3,a        ;Zu den LED's ausgeben
 sjmp Start      ;Endlos wiederholen
```

Erzeugte Prog1.lst:

```
                                 1 ;*****************************
                                 2 ;* Testprogramm für Assembler *
                                 3 ;*****************************
                                 4 .area HOME (ABS)
004000                           5 .org 0x4000
                                 6
004000                           7 main:
004000 E5 90            [12]     8  mov a,P1  ;Schalterstellung einlesen
004002 F5 B0            [12]     9  mov P3,a  ;Zu den LED's ausgeben
004004 80 FA            [24]    10  sjmp main ;Endlos wiederholen
```

4.3.1 Mehrere ASM-Quelldateien

Auch ein Assemblerprogramm kann aus mehreren Teilprogrammen bestehen. Dabei sind die Teilprogramme einzeln zu assemblieren und die .rel Dateien anschließend zu einem lauffähigen Programm zu linken.

Teilprogramm Prog1Proz.asm:

```
;****************************************
;* Assembler Unterprogramm für Prog1.asm *
;****************************************/
.globl Timer

Timer:
 mov R7,#0xFF    ;Zähler für Wartezeit
 djnz r7,$       ;herunterzählen
 ret             ;Zurück ins Hauptprogramm
```

Assemblieren mit: sdas8051 -plosgff

Hauptprogramm Prog1.asm:

```
;*****************************
;* Testprogramm für Assembler *
;*****************************
.area HOME (ABS)
.org 0

Start:
 mov a,P1        ;Schalterstellung einlesen
 mov P3,a        ;Zu den LED's ausgeben
 lcall Timer     ;Externes Unterprogramm
 sjmp Start      ;Endlos wiederholen
```

Assemblieren mit: sdas8051 -plosgff

Linken durch: sdcc -o %p/ %p/%n.rel %p/Prog1Proz.rel

Resultierende Prog1.ihx:

```
:09000000E590F5B012002080F734
:050020007FFFDFDC2280
:00000001FF
```

4.4 C und ASM Programm gemischt

Da die Größe und das Timing durch Assemblerprogramme exakt festgelegt werden kann, ist es manchmal sinnvoll ein Teilprogramme in Assembler in einem C-Programm zu verwenden.

Assembler Unterprogramm Prog1Proz.asm:

```
;*************************************
;* Assembler Unterprogramm für Prog1.c *
;*************************************/
.globl Timer

Timer:
 mov R7,#0xFF    ;Zähler für Wartezeit
 djnz r7,$       ;herunterzählen
 ret             ;Zurück ins Hauptprogramm
```

Assemblieren mit: sdas8051 -plosgff

C-Hauptprogramm Prog1.c:

```
/*****************************
 * Testprogramm für C-Compiler *
 *****************************/
#include <8051.h>

//--- Externe Funktionen -----------------------------
extern void Timer(void);

int main ()
{ char cVar;
  for(;;)        //Endlosschleife
  { cVar = P1;   //Schalterstellung einlesen
     P3 = cVar; //Zu den LED's ausgeben
     Timer();   //Externe Prozedur aufrufen
  }
}
```

Kompilieren durch: sdcc -mmcs51 --debug --verbose --nolospre -c -o %p/ %c

Linken durch: sdcc -o %p/ %p/%n.rel %p/Prog1Proz.rel

4.4.1 C-Programm mit ASM-Startup

Manchmal ist es nötig für ein Programm gezielte Voreinstellungen und Parameter vorzugeben, dies läßt sich komfortabel mit einer Startup-Datei in Assembler durchführen. Im Beispiel residiert im Speicherbereich von 0000h bis 0x3FFFh ein Monitorprogramm. Dieses Programm bedient den Download von Programmen in den Arbeitsspeicher ab 4000h und stellt den Anwenderprogrammen eine ganze Reihe von Unterprogrammen zur Verfügung. Aufgabe der Startupdatei ist nun die Speicherverwaltung, das bereitstellen von Systemvariablen, die Verwaltung der Interruptvektoren und das deklarieren der Prozeduradressen für das Anwenderprogramm.

Beispiel Startupdatei Startup.asd:

```
;*******************************************
;* IS53 - Startup-Datei für C-Programme     *
;* Für Monitorprogramm V1.30 (C) April 2018  *
;*******************************************
BaseAdr == 0x4000           ; Basisadresse Arbeitsspeicher
MMBase  == 0x00F0           ; MemoryMap Basisadresse

.area HOME (ABS)

_startup::
 ljmp ISPSTART         ; Autostart-Programm
; ajmp ISPSTART        ; Fremdstart-Programm
; nop

;-----------------------------------------------------------------
;* Benutzte Interrupts durch entfernen des ";" freigeben
; LJMP _INT0ISR                  ;INT0 Vektor
 RETI
 MOV a,#1
; LJMP     _TF0ISR               ;TF0
 RETI
 MOV a,#1
; LJMP     _INT1ISR              ;INT1
 RETI
 MOV a,#1
; LJMP     _TF1ISR               ;TF1
 RETI
 MOV a,#1
; LJMP     _RI_TIISR             ;RI/TI
 RETI
 MOV a,#1
; LJMP     _SerialInterrupt      ;RI/TI
 RETI
 MOV a,#1
; LJMP     _T2ISR                ;T2
 RETI
 MOV a,#1
; LJMP     _INT6ISR              ;Interrupt 6
 RETI
 MOV a,#1
; LJMP     _INT7ISR              ;Interrupt 7
 RETI
 MOV a,#1
```

```
; LJMP     _INT8ISR              ;Interrupt 8
 RETI
 MOV a,#1
; LJMP     _INT9ISR              ;Interrupt 9
 RETI
 MOV a,#1
; LJMP     _INT10ISR             ;Interrupt 10
 RETI
 MOV a,#1
; LJMP     _INT11ISR             ;Interrupt 11
 RETI
 MOV a,#1
;LJMP _INT12ISR                  ;Interrupt 12
 RETI
 MOV a,#1
; LJMP     _INT13ISR             ;Interrupt 13
 RETI
 MOV a,#1
; LJMP     _INT14ISR             ;Interrupt 14
 RETI
 MOV a,#1

;.DS  BaseAdr + 0x100 - .    ;Reservieren bis X100h
.DS   0xCA  ;Reservieren bis X100h
ISPSTART:       MOV     SP,#0x2F  ;Stack einrichten bei 0x30

;---------------------------------------------------------------------
; Deklaration der Unterprogrammadressen im ROM

; Ausgabefunktionen
_OPLA1     == 0x0F36
_OPLA1BS   == 0x17E9
_OPLA1BR   == 0x1804
_OPLA1BC   == 0x1822
_SEROA     == 0x16C6
_SERAUSG   == 0x1712
_SERPBS    == 0x1841
_SERPBR    == 0x185F
_SERPBC    == 0x187F
_CLRLD1    == 0x1923
_CLRLD2    == 0x191D
_SETLD1    == 0x1901
_SETLD2    == 0x18FB

; Eingabefunktionen
_IPLA1     == 0x0F52
_IPLA1BIT  == 0x189C
_SERIA     == 0x16EE
_SEREING   == 0x174C
_EWIBIT    == 0x18BC

;serielle Schnittstelle
_SIOABLOCK == 0x129B
_SIODBLOCK == 0x1276
_CHRDTO1   == 0x12BE
_CHRDTO2   == 0x12EE
_BYTERD    == 0x0E66
```

```
; Zeitfunktionen
_MINH      == 0x0F07
_SECH      == 0x0EB1
_ZSECH     == 0x0E95
_HSECH     == 0x0ED7
_MSECH     == 0x0EF3
_T0PAUSE   == 0x0B3D
_INT0PAUSE == 0x0B44

;Codewandlung
_ASCHEX    == 0x0E00
_HEXASC    == 0x0E37
_HEXDEZ    == 0x0F74
_HEX16DEZ  == 0x0FA5
_DEZHEX    == 0x10B3
_DEZ16HEX  == 0x111B
_BCH1TO7   == 0x1320
_BCH2TO7   == 0x1349

;Arithmetik
_DIV16     == 0x11A1
_DIV32     == 0x13D6
_MUL16     == 0x1373
_CMP16     == 0x124A

;Programmbeendigung
_CLRAUTO   == 0x135F
_TASTEND   == 0x18D8
_T0END     == 0x0F5C
_T1END     == 0x0F68
_INT0END   == 0x0F6C
_INT1END   == 0x0F70

;Befehlserweiterung
_PUSHREG   == 0x1679
_POPREG    == 0x1642
_XCHREG    == 0x15D0
_PUSHDPTR  == 0x1555
_POPDPTR   == 0x1574
_XCHDPTR   == 0x1589
_DECDPTR   == 0x15B6

;Sonstige
_START_OS  == 0x0000

;--- Für Hardware-Debugging --------------
;SERIALINTERRUPT CODE  193FH
;_GETKEY         CODE  1B5DH
;_ISD_CHAR       CODE  0008H
;_ISKEY          CODE  1B7DH
;_PUTCHAR        CODE  1B32H
;__ISD_INIT      CODE  1B97H
;?ISD?READSFR00  CODE  1A02H
;?ISD?READSFR01  CODE  1A12H

.area XVARS (ABS)
```

```
;--- XDATA-Variablen ------------------------------------
STARTFLAG  == BaseAdr       ; Startkennung
EP0        == MMBase + 8          ; Expanderport 0
EP1        == MMBase + 9          ; Expanderport 1
EP2        == MMBase + 10         ; Expanderport 2
EP3        == MMBase + 11         ; Expanderport 3
EP4        == MMBase + 12         ; Expanderport 4
EP5        == MMBase + 13         ; Expanderport 5
EP6        == MMBase + 14         ; Expanderport 6
EP7        == MMBase + 15         ; Expanderport 7

EOUTPORT1  == BaseAdr + 0x30      ; Rettungszellen für Ausgabeerweiterung
EOUTPORT2  == BaseAdr + 0x31
EOUTPORT3  == BaseAdr + 0x32
EOUTPORT4  == BaseAdr + 0x33
EOUTPORT5  == BaseAdr + 0x34
EOUTPORT6  == BaseAdr + 0x35
EOUTPORT7  == BaseAdr + 0x36
EOUTPORT8  == BaseAdr + 0x37
EINPORT1   == BaseAdr + 0x38      ; Rettungszellen für Eingabeerweiterung
EINPORT2   == BaseAdr + 0x39
EINPORT3   == BaseAdr + 0x3A
EINPORT4   == BaseAdr + 0x3B
EINPORT5   == BaseAdr + 0x3C
EINPORT6   == BaseAdr + 0x3D
EINPORT7   == BaseAdr + 0x3E
EINPORT8   == BaseAdr + 0x3F

RETR0      == BaseAdr + 0x70      ; Rettungszelle Register 0
RETR1      == BaseAdr + 0x71      ; Rettungszelle Register 1
RETR2      == BaseAdr + 0x72      ; Rettungszelle Register 2
RETR3      == BaseAdr + 0x73      ; Rettungszelle Register 3
RETR4      == BaseAdr + 0x74      ; Rettungszelle Register 4
RETR5      == BaseAdr + 0x75      ; Rettungszelle Register 5
RETR6      == BaseAdr + 0x76      ; Rettungszelle Register 6
RETR7      == BaseAdr + 0x77      ; Rettungszelle Register 7
RETDPL     == BaseAdr + 0x78      ; Rettungszelle Datenpointer L
RETDPH     == BaseAdr + 0x79      ; Rettungszelle Datenpointer H
RETPSW     == BaseAdr + 0x7A      ; Rettungszelle Prozessor-Status-Wort
RETACC     == BaseAdr + 0x7B      ; Rettungszelle Akku
RETB       == BaseAdr + 0x7C      ; Rettungszelle Register B
```

Das Hauptprogramm ist ein tabellengesteuertes Lauflicht an den LED‘s eines Ausgabeports und bedient sich für die Ausgabe und Zeitverzögerung bei den Unterprogrammen des Monitorprogramms.

Datei IS53XTLL.c:

```
/*********************************
* Testprogramm für SDCC-Compiler  *
* mit Hardware IS53               *
*********************************/
#pragma noiv
#include <at89s8253.h>
```

```
typedef unsigned char BYTE;
typedef unsigned int WORD;

//--- Externe Prozeduren ---------------------------
extern void OPLA1(BYTE bOut);
extern void ZSECH(BYTE bTime);

//--- Tabelle der Ausgabewerte ---------------------
__code BYTE bTab[] = { 0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,
     0x40,0x20,0x10,0x08,0x04,0x02 };

void main(void)
{ BYTE bCount, bVar;
  __xdata __at (0x4000) BYTE bASFlag;
  bASFlag = 0x01;      // Autostart löschen

  bCount = 0;          // Offset = 1. Ausgabewert
  for(;;)              // Programmschleife
  { bVar = (bTab[bCount++]);  // Tabellenwert holen
    OPLA1(bVar);       // Wert ausgeben
    if(bCount > 13)    // Tabellenende ?
      BCount = 0;      // Offset = 1. Ausgabewert
    ZSECH(10);         // 10 Zehntelsekunden warten (1s.)
  }
}
//--- Interrupt-Service-Routinen -------------------------------
//--- Benutzte Interrupts durch entfernen von "//" freigeben ----
//void INT0ISR (void) __interrupt(0) __using(1) { }     // INT0 Interrupt
//void TF0ISR  (void) __interrupt(1) __using(1) { }     // TF0 Interrupt
//void INT1ISR (void) __interrupt(2) __using(1) { }     // INT1 Interrupt
//void TF1ISR  (void) __interrupt(3) __using(1) { }     // TF1 Interrupt
//void RI_TIISR(void) __interrupt(4) __using(1) { } //RI/TI Interrupt (UART)
//void T2ISR   (void) __interrupt(5) __using(1) { }     // T2 Interrupt
```

Da die Aufrufe für assemblieren, kompilieren und linken recht aufwändig werden können, kann man diese vorteilhaft in einem Makefile verpacken und dadurch mit einem einzigen Aufruf desselben die ganze Arbeit erledigen.

4.5 Makefile

Beispiel für ein SDCC-Makefile:

Hier werden die C-Datei „IS53XTLL.C“ und die Assemblerdatei „Startup.asd“ übersetzt und zu IS53XTLL.ihx gelinkt. Dabei wird als Speicheradresse 0x4000 (in Startup.asd) und als Speichermodel small vorgegeben. Der Speicher des C-Programms beginnt bei 0x4100.

Auszug Startup.asd:

```
      BaseAdr    == 0x4000        ; Basisadresse Arbeitsspeicher
      MMBase     == 0x00F0        ; MemoryMap Basisadresse

      .area HOME (ABS)
      .org BaseAdr
      _startup::
       ljmp ISPSTART        ; Autostart-Programm
```

Beispiel Makefile:

```
#########################################################
# GNU Makefile demonstrating combination of C and assembly source files
# File Name: makefile
# All targets in the makefile are processed sequentially by SDCC
# To create the file ‘file.hex’ using GNU make, just execute ‘make’
#########################################################
# Hier sind zusätzliche Pfade für Include Dateien
#INCLUDES := -I”C:\WeissNichtWas\”

#Hier wird das Speichermodel gewählt
#MODEL = --model-large
MODEL = --model-small

#Hier werden Aufrufparameter für SDCC festgelegt
CFLAGS := -c --debug $(MODEL) $(INCLUDES)

#Hier werden Aufrufparameter für den Linker festgelegt
LFLAGS := --debug --code-loc 0x4100 --code-size 0xC000 --xram-loc 0x0000 --xram-
size 0x4000 $(MODEL)

#Hier wird das Default Target festgelegt
all: Datei.hex

#Hier wird festgelegt, welche Dateien zusammen zur Objekt Datei gelinkt werden.
OBJECTS := IS53XTLL.rel Startup.rel

#Hier sind die Regeln damit der Linker eine ihx Datei erzeugt die anschließend,
#durch packihx gewandelt werden kann
file.hex : $(OBJECTS)
      sdcc $(LFLAGS) $^
#     packihx IS53XTLL.ihx > IS53XTLL.hex

#Hier wird jede C-Datei an den Preprozessor gegeben um eine asm-Datei zu erzeugen
#und daraus die .rel-Datei zu assemblieren.
%.rel : %.c
      sdcc $(CFLAGS) $<

#Hier  werden  die  nicht  von  SDCC  erzeugten  Assemblerdateien  zu  .rel-Dateien
assembliert.
%.rel : %.asd
      sdas8051 -plosgff $<

#Hier werden die nicht mehr benötigten Zwischendateien gelöscht.
# Hinweis: zum löschen der Dateien "make clean" aufrufen.
clean:
      $(RM) *.lst $(RM) %.asm $(RM) *.bak
      $(RM) *.sym $(RM) *.map $(RM) *.omf $(RM) *.adb
      $(RM) *.cdb $(RM) *.mem $(RM) *.rel $(RM) *.lk
```

Packihx ist deaktiviert, da die Zielhardware direkt mit .ihx zurechtkommt.

Der Bereich „clean“ wird nicht automatisch aufgerufen, er muss durch das Kommando „Make clean“ gestartet werden. Im ProgEdi ist dafür im Menü – Erstellen der Item „Make Clean“ vorgesehen.

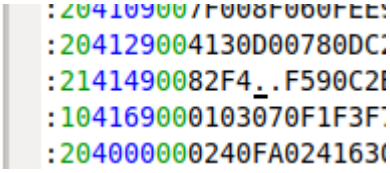
4.6 Intel-Hex Datei

Wird, im Hexmode, der Cursor im Datenfeld bewegt, zeigt die Statuszeile links die Speicheradresse des aktuellen Byte als z.B:

Byteadresse: 0x40FC

Die zusammengehörigen zwei Zeichen des Datenbyte beim Cursor sind durch einen roten Unterstrich markiert. Befindet sich der Cursor auf der Anzahl der Recordbyte, wird diese dezimal in der Statuszeile angezeigt.

Es ist möglich den Inhalt zu editieren, wobei die Prüfsumme automatisch mitgeführt wird. Die +-Taste fügt ein Byte in den Record ein. Es wird dann .. gezeigt und die Punkte sind mit den Zeichen des Byte zu füllen. Anzahl und Prüfsumme passen sich automatisch an.



Im Überschreibmodus werden die bestehenden Werte überschrieben, wobei die Anzahl und Adresse der Werte gleich bleibt. Auch hier wird die Prüfsumme automatisch aktualisiert.

Im Textmode wird eine Intel-Hex Datei behandelt, wie jede andere Textdatei.

5 Konfigurationsdatei

Viele der Funktionen des Editors sind von den Einstellungen in der Konfigurationsdatei abhängig, deshalb hier ein Blick auf den Inhalt dieser Datei. Da sie eine einfache Textdatei ist, kann hier alles mit einem Editor verändert werden. Zum editieren dieser Datei empfiehlt es sich, einen anderen Editor zu verwenden, als den ProgEdi. Denn sonst könnten während des laufenden Betriebs dringend benötigte Parameter geändert sein, was zu nicht vorhersehbaren Ergebnissen führt.

5.1 [fenster]

Hier ist die Fensterbreite (FBreite) und Fensterhöhe (FHoehe) beim Beenden des Programms hinterlegt. Sie wird bei einem erneuten Start des Editors auf diese Werte neu eingestellt.

5.2 [zielsystem]

Hier finden sich Einstellungen für die Hardware des Zielsystems. Wird derzeit noch nicht genutzt.

5.3 [editor]

CurrentTab: Nummer der aktiven Datei im Editor, Start mit 0.
CurrentPos: Cursorposition im Text
LetzterPfad: Pfad der zuletzt aktiven Datei, ohne Dateiname

5.4 [SDCC_Com]

Kommandos für das Menü Erstellen.

C_Com: Kommando zum Aufruf des C-Compilers
ASM_Com: Kommando zum Aufruf des Assembler
LINK_Com: Kommando zum Aufruf des Linkers
Down_Com: Kommando zum Aufruf des Download
Send_Com: Kommando zum Aufruf von Senden
Make_Com: Pfad zum Makefile

5.5 [syntaxcolor]

Vorder(F)- und Hinter(B)-grundfarben der Textelemente, welche Scintilla vorgibt, mit verkürzten Namen. Bei einigen ist klar wofür sie zuständig sind, bei anderen nicht da sie in meinen Programmen nicht vorkommen. Die Unklaren sind mit einem Fragezeichen „?“ gekennzeichnet.
Bei Wortlisten ist darauf zu achten, die eindeutigeren Worte zuerst aufzuführen. Bei ADDC und ADD ist das ADDC eindeutiger, umgekehrt wird sonst das Teilwort ADD zuerst als Wort erkannt.

5.5.1 C-Syntax

C_Instruction: Liste der Schlüsselworte für C (wxSTC_C_WORD)
auto break case continue default do else enum extern for goto if inline register restrict return char const double float int long short signed static union unsigned sizeof struct switch void volatile while
C_InstFCol: Vordergrundfarbe der Schlüsselwörter
C_InstBCol: Hintergrundfarbe der Schlüsselwörter
z.B: F=rgb(0, 0, 255), B=rgb(255, 255, 255)

C_Register: Liste der Registerbezeichnungen für C-51 (wxSTC_C_WORD2)
P0 SP DPL DPH PCON TCON TMOD TL0 TL1 TH0 TH1 P1 SCON SBUF P2 IE P3 IP T2CON T2MOD RCAP2L RCAP2H TL2 TH2 PSW ACC B
C_RegiFCol: Vordergrundfarbe der Register
C_RegiBCol: Hintergrundfarbe der Register
z.B: F=rgb(160,32,240), B=rgb(255, 255, 255)

C_DefaFCol: Default Vordergrundfarbe (wxSTC_C_DEFAULT)
C_DefaBCol: Default Hintergrungfarbe
z.B: F=rgb(0, 0, 0), B=rgb(255, 255, 255)
Normaler Text

C_CommFCol: Vordergrund Kommentar (wxSTC_C_COMMENT)
C_CommBCol: Hintergrund Kommentar
z.B: F=rgb(0, 150, 0), B=rgb(255, 255, 255)
//Kommentar

C_CoLnFCol: Vordergrund Kommentarzeile (wxSTC_C_COMMENTLINE)
C_CoLnBCol: Hintergrund Kommentarzeile
z.B: F=rgb(0, 150, 0), B=rgb(255, 255, 255)
/* Dies ist ein Test ….

C_CoDoFCol: Vordergrund mehrzeiliger Kommentar (wxSTC_C_COMMENTLINEDOC)
C_CoDoBCol: Hintergrund mehrzeiliger Kommentar
z.B: F=rgb(0, 150, 0), B=rgb(255, 255, 255)
```
/*
  Test
*/
```

C_NumbFCol: Vordergrundfarbe für Zahlen (wxSTC_C_NUMBER)
C_NumbBCol: Hintergrundfarbe für Zahlen
z.B: F=rgb(255, 0, 0), B=rgb(255, 255, 255)
0x3B 12345678 7Ch

C_CstrFCol: Vordergrundfarbe für C_String (wxSTC_C_STRING)
C_CstrBCol: Hintergrundfarbe für C_String
z.B: F=rgb(0, 100, 0), B=rgb(255, 255, 255)
„Test“

C_CharFCol: Vordergrundfarbe für C_Char (wxSTC_C_CHARACTER)
C_CharBCol: Hintergrundfarbe für C_Char
z.B: F=rgb(0, 100, 0), B=rgb(255, 255, 255)
‚T‘

C_UuidFCol: Vordergrundfarbe ? (wxSTC_C_UUID)
C_UuidBCol: Hintergrundfarbe ?
z.B: F=rgb(255, 255, 0), B=rgb(255, 255, 255)
Test

C_PrepFCol: Vordergrundfarbe für Preprozessor (wxSTC_C_PREPROCESSOR)
C_PrepBCol: Hintergrundfarbe für Preprozessor
z.B: F=rgb(139, 69, 19), B=rgb(255, 255, 255)
#include <xxxxx.h>

C_OperFCol: Vordergrundfarbe für Operator (wxSTC_C_OPERATOR)
C_OperBCol: Hintergrundfarbe für Operator
z.B: F=rgb(0, 0, 0), B=rgb(255, 255, 255)
Test

C_IdenFCol: Vordergrundfarbe für Identifizierer (wxSTC_C_IDENTIFIER)
C_IdenBCol: Hintergrundfarbe für Identifizierer
z.B: F=rgb(0, 0, 0), B=rgb(255, 255, 255)
typedef BYTE INT0ISR

C_SeolFCol: Vordergrundfarbe offener String (wxSTC_C_STRINGEOL)
C_SeolBCol: Hintergrundfarbe offener String
z.B: F=rgb(255, 0, 0), B=rgb(255, 220, 200)
„Test ,Test

C_VerbFCol: Vordergrundfarbe ? (wxSTC_C_VERBATIM)
C_VerbBCol: Hintergrundfarbe ?
z.B: F=rgb(160, 32, 240), B=rgb(255, 255, 255)
Test

C_RegeFCol: Vordergrundfarbe für ? (wxSTC_C_REGEX)
C_RegeBCol: Hintergrundfarbe für ?
z.B: F=rgb(0, 0, 0), B=rgb(255, 255, 255)
Test

C_CldoFCol: Vordergrundfarbe für ? (wxSTC_C_COMMENTLINEDOC)
C_CldoBCol: Hintergrundfarbe für ?
z.B: F=rgb(0, 0, 0), B=rgb(255, 255, 255)
Test

C_CdkwFCol: Vordergrundfarbe für ? (wxSTC_C_COMMENTDOCKEYWORD)
C_CdkwBCol: Hintergrundfarbe für ?
z.B: F=rgb(139, 117, 0), B=rgb(255, 255, 255)
Test

C_CdkeFCol: Vordergrundfarbe für ? (wxSTC_C_COMMENTDOCKEYWORDERROR)
C_CdkeBCol: Hintergrundfarbe für ?
z.B: F=rgb(139, 117, 0), B=rgb(255, 255, 255)
Test

C_SingleBit: Liste der Einzelbitbezeichner für C-51
EXF2 EXN2 AC CY F0 RS1 RS0 OV IT0 IE0 IT1 IE1 TR0 TF0 TR1 TF1 T2 T2EX RI TI RB8 TB8 REN SM2 SM1 SM0 EX0 ET0 EX1 ET1 ES ET2 EA rxd TXD INT0 INT1 T0 T1 WR RD PX0 PT0 PX1 PT1 PS PT2 CP_R C_T TR2 tclk RCLK TF2 P0_0 P0_1 P0_2 P0_3 P0_4 P0_5 P0_6 P0_7 P1_0 P1_1 P1_2 P1_3 P1_4 P1_5 P1_6 P1_7 P2_0 P2_1 P2_2 P2_3 P2_4 P2_5 P2_6 P2_7 P3_0 P3_1 P3_2 P3_3 P3_4 P3_5 P3_6 P3_7

C_GlobFCol: Vordergrundfarbe für Einzelbit (wxSTC_C_GLOBALCLASS)
C_GlobBCol: Hintergrundfarbe für Einzelbit
z.B: F=rgb(255, 0, 255), B=rgb(255, 255, 255)

C_StrrFCol: Vordergrundfarbe für ? (wxSTC_C_STRINGRAW)
C_StrrBCol: Hintergrundfarbe für ?
z.B: F=rgb(139, 117, 0), B=rgb(255, 255, 255)
Test

C_TrivFCol: Vordergrundfarbe für ? (wxSTC_C_TRIPLEVERBATIM)
C_TrivBCol: Hintergrundfarbe für ?
z.B: F=rgb(139, 117, 0), B=rgb(255, 255, 255)
Test

C_HashFCol: Vordergrundfarbe für ? (wxSTC_C_HASHQUOTEDSTRING)
C_HashBCol: Hintergrundfarbe für ?
z.B: F=rgb(139, 117, 0), B=rgb(255, 255, 255)
Test

C_PpcoFCol: Vordergrundfarbe für ? (wxSTC_C_PREPROCESSORCOMMENT)
C_PpcoBCol: Hintergrundfarbe für ?
z.B: F=rgb(139, 117, 0), B=rgb(255, 255, 255)
Test

C_UsliFCol: Vordergrundfarbe für ?
C_UsliBCol: Hintergrundfarbe für ?
z.B: F=rgb(139, 117, 0), B=rgb(255, 255, 255)
Test

#C_TaskFCol: Vordergrundfarbe für Task marker and error marker keywords
C_TaskBCol: Hintergrundfarbe für Task marker and error marker keywords
z.B: F=rgb(0, 0, 0), B=rgb(255, 255, 255)
Test Nur innerhalb eines Kommentars, nur Hintergrund möglich

C_EscaFCol: Vordergrundfarbe für ?
C_EscaBCol: Hintergrundfarbe für ?
z.B: F=rgb(139, 117, 0), B=rgb(255, 255, 255)
Test

5.5.2 ASM-Syntax

ASM_Instruction: Liste der mnemonischen CPU-Befehle
acall addc add ajmp anl cjne clr cpl da dec div djnz inc jbc jb jc jmp jnb jnc jnz jz lcall ljmp movc movx mov mul nop orl pop push reti ret rlc rl rrc rr setb sjmp subb swap xchd xch xrl
ASM_InstFCol: Vordergrundfarbe für Befehle (wxSTC_ASM_CPUINSTRUCTION)
ASM_InstBCol: Hintergrundfarbe für Befehle
z.B: F=rgb(0, 0, 255), B=rgb(255, 255, 255)

ASM_Register: Liste der CPU-Register
p0 sp dpl dph pcon tcon dpl dph tcon tmod tl0 tl1 th0 th1 p1 scon sbuf p2 ie p3 ip t2con t2mod rcap2l rcap2h tl2 th2 psw acc regb
ASM_RegFCol: Vordergrundfarbe für Register (wxSTC_ASM_REGISTER)
ASM_RegBCol: Hintergrundfarbe für Register
z.B: F=rgb(160,32,240), B=rgb(255, 255, 255)

ASM_ExtInst: Liste erweiterter Befehle

.area .equ .end .globl .org @a ab dptr pc r0 r1 r2 r3 r4 r5 r6 r7

ASM_ExtrFCol: Vordergrundfarbe für erweiterte Befehle (wxSTC_ASM_EXTINSTRUCTION)
ASM_ExtrBCol: Hintergrundfarbe für erweiterte Befehle
z.B: F=rgb(139, 69, 19), B=rgb(255, 255, 255)

ASM_CommFCol: Vordergrundfarbe für Kommentar (wxSTC_ASM_COMMENT)
ASM_CommBCol: Hintergrundfarbe für Kommentar
z.B: F=rgb(0, 150, 0), B=rgb(255, 255, 255)
;Kommentar

ASM_DefaFCol: Vordergrundfarbe für Default (wxSTC_ASM_DEFAULT)
ASM_DefaBCol: Hintergrundfarbe für Default
z.B: F=rgb(0, 0, 0), B=rgb(255, 255, 255)
Default

ASM_NumbFCol: Vordergrundfarbe für Zahlen (wxSTC_ASM_NUMBER)
ASM_NumbBCol: Hintergrundfarbe für Zahlen
z.B: F=rgb(255, 0, 0), B=rgb(255, 255, 255)
1234 0x25 7Ch

ASM_StriFCol: Vordergrundfarbe für String (wxSTC_ASM_STRING)
ASM_StriBCol: Hintergrundfarbe für String
z.B: F=rgb(0, 255, 0), B=rgb(255, 255, 255)
„Test“

ASM_OperFCol: Vordergrundfarbe für Operator (wxSTC_ASM_OPERATOR)
ASM_OperBCol: Hintergrundfarbe für Operator
z.B: F=rgb(0, 0, 0), B=rgb(255, 255, 255)
Test

ASM_IdenFCol: Vordergrundfarbe für Identifizierer (wxSTC_ASM_IDENTIFIER)
ASM_IdenBCol: Hintergrundfarbe für Identifizierer
z.B: F=rgb(0, 0, 0), B=rgb(255, 255, 255)
Test

ASM_SingleBit: Liste der Einzelbit-Bezeichner
ac cy f0 rs1 rs0 ov it0 it1 ie0 ie1 tr0 tr1 tf0 tf1 t2ex t2 ri ti rb8 tb8 ren sm2 sm1 sm0 ex0 ex1 et0 et1 es et2 ea rxd txd int0 int1 t0 t1 wr rd px0 px1 pt0 pt1 ps pt2 cp_r c_t tr2 exn2 tclk rclk exf2 tf2 P0_0 P0_1 P0_2 P0_3 P0_4 P0_5 P0_6 P0_7 P1_0 P1_1 P1_2 P1_3 P1_4 P1_5 P1_6 P1_7 P2_0 P2_1 P2_2 P2_3 P2_4 P2_5 P2_6 P2_7 P3_0 P3_1 P3_2 P3_3 P3_4 P3_5 P3_6 P3_7
ASM_MathFCol: Vordergrundfarbe für Einzelbit (wxSTC_ASM_MATHINSTRUCTION)
ASM_MathBCol: Hintergrundfarbe für Einzelbit
z.B: F=rgb(255, 0, 255), B=rgb(255, 255, 255)

ASM_DireFCol: Vordergrundfarbe für Direktive (wxSTC_ASM_DIRECTIVE)
ASM_DireBCol: Hintergrundfarbe für Direktive
z.B: F=rgb(255, 255, 0), B=rgb(255, 255, 255)
Test

ASM_DiOpFCol: Vordergrundfarbe für ? (wxSTC_ASM_DIRECTIVEOPERAND)
ASM_DiOpBCol: Hintergrundfarbe für ?
z.B: F=rgb(139, 117, 0), B=rgb(255, 255, 255)
Test

ASM_CoBlFCol: Vordergrundfarbe für ? (wxSTC_ASM_COMMENTBLOCK)
ASM_CoBlBCol: Hintergrundfarbe für ?

z.B: F=rgb(0, 100, 0), B=rgb(255, 255, 255)
Test

ASM_CharFCol: Vordergrundfarbe für Zeichen (wxSTC_ASM_CHARACTER)
ASM_CharBCol: Hintergrundfarbe für Zeichen
z.B: F=rgb(160, 32, 240), B=rgb(255, 255, 255)
‚T' ‚x' ‚7' ‚Test' ‚Hallo Welt!'

ASM_CoDiFCol: Vordergrundfarbe für ? (wxSTC_ASM_COMMENTDIRECTIVE)
ASM_CoDiBCol: Hintergrundfarbe für ?
z.B: F=rgb(30, 144, 255), B=rgb(255, 255, 255)
Test

5.6 [dateien]

Die beim beenden des Editors offenen Dateien. „AFile" ist die Datei ganz links im Editor-Notebook, „BFile" die Datei rechts daneben ..... usw. Sie werden beim nächsten Start in dieser Reihenfolge wieder geöffnet und dargestellt.

5.7 [lesezeichen]

Die Lesezeichen der beim beenden offenen Dateien. Die erste Zahl ist die Anzahl aller Lesezeichen im Text, die folgenden Zahlen sind die Nummern der Zeilen mit Lesezeichen.

ALesez=4 12 24 74 98

Bedeutet: im AFile (linke Datei) befinden sich vier Lesezeichen, in den Zeilen 12, 24, 74 und 98. Die Lesezeichen werden beim Programmstart für jede Datei wieder hergestellt. Enthält eine geöffnete Datei kein Lesezeichen, ist die Anzahl 0 (z.B: ALesez=0).

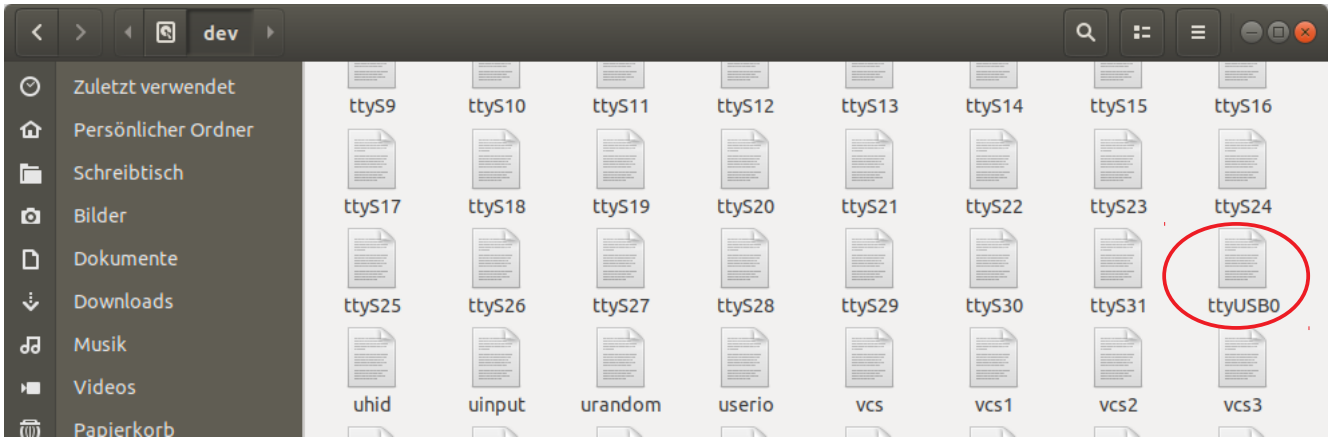
6 USB unter Linux

Unter Linux ist es nicht ganz einfach eine zugekaufte oder selbstgebaute Hardware einzubinden, wenn sie nicht automatisch vom System erkannt wird. Dies betrifft auch die vielen verschiedenen Programmieradapter für Mikrocontroller.

6.1 USB-RS232 Wandler

Die RS232-Schnittstelle ist eine genormte serielle Schnittstelle, welche mit unterschiedlichen Bezeichnungen verbreitet ist. Nach amerikanischer Norm heißt sie RS232, nach europäischer Norm V.24 und nach deutscher Norm DIN 66020. Da heute die meisten PC‘s keine RS232-Schnittstelle mehr haben, werden billige USB-RS232 Adapter angeboten. Die Hersteller dieser Adapter scheren sich nicht viel um die Norm, was die Sache etwas verkompliziert. Doch das ist im Netz vielfach beschrieben und wird hier nicht weiter ausgeführt. Hier geht es um das einbinden und benutzen dieser Adapter.

Beim einstecken eines funktionierenden Adapter an einen USB-Anschluß, wird er vom System erkannt und unter einem bestimmten Namen eingetragen. Diesen Namen herauszufinden ist wichtig, um die Schnittstelle ansprechen zu können. Dazu sieht man sich im Hauptverzeichnis der Festplatte den Inhalt des Ordners „dev“ (devices - Geräte) genauer an.



Der hier verwendete Adapter wurde unter dem Namen ttyUSB0 eingetragen, wobei die Nummer 0 eine fortlaufende Nummer ist. Wird zusätzlich ein zweiter, gleicher Adapter angesteckt erhält er die nächst höhere Nummer, also ttyUSB1. Ein Teil meiner Adapter meldet sich mit ttyACMx (Abstract Control Model). Um sicherzustellen dass es auch wirklich der richtige Adapter ist, kann man ihn wieder abstecken und der Eintrag verschwindet.

ttyUSBx = USB-UART Konverter (Hardware), ttyACMx = USB-CDC Programm (Software).

Die Schnittstellen sind auch durch „dmesg“ in einem Terminal ermittelbar:

```
dmesg | grep tty
```

```
tom@tom-TM6595T:~/LDaten$ dmesg | grep tty
[    0.000000] console [tty0] enabled
[    0.708179] 0000:00:16.3: ttyS4 at I/O 0x30b0 (irq = 19, base_baud = 115200) is a 16550A
[27540.663961] cdc_acm 2-1.2:1.0: ttyACM0: USB ACM device
```

Damit ist nun bekannt unter welchem Namen der Adapter angesprochen wird.

Nun ist zu prüfen, ob man die Berechtigung besitzt auf das Gerät zugreifen zu dürfen. Dazu in der Konsole die Gruppenzugehörigkeit ermitteln mit "id":

```
tom@tom-TM6595T:~/LDaten/QtProgs$ id
uid=1000(tom) gid=1000(tom) Gruppen=1000(tom),4(adm),20(dialout),24(cdrom),
27(sudo),30(dip),46(plugdev),118(lpadmin),128(sambashare)
```

Um mit den seriellen Schnittstellen arbeiten zu dürfen muss man sich in der Gruppe „dialout“, und für die USB-Schnittstelle besser auch in der Gruppe „plugdev“ befinden. In meinem Fall ist das hier gegeben, in der Liste sind die Gruppen 20(dialout) und 46(plugdev aufgeführt. Ist man kein Mitglied der entsprechenden Gruppe, muss man sich dieser anschließen:

Bestehenden Benutzer einer weiteren Gruppe hinzufügen:

Allgemein: sudo usermod -aG GRUPPENNAME BENUTZERNAME

Beispiel: sudo usermod -aG plugdev tom

Als Benutzername ist nicht tom, sondern der Eigene einzusetzen.

Nachdem nun der Zugriff auf die Hardware sichergestellt ist, wird jetzt die Software für das Downloaden eingerichtet. Dazu sind die Übertragungsparameter der Zielhardware richtig einzustellen, damit sich die Teilnehmer untereinander verstehen. Welche die richtigen Parameter der Zielhardware sind, ist deren Dokumentation zu entnehmen. Die Parameter für den PC und damit für den ProgEdi werden mit dem „stty“ Kommando bestimmt:

```
stty 9600 -F /dev/ttyUSB0
```

Hier wird die Baudrate von ttyUSB0 auf 9600 Bit je Sekunde eingestellt. Näheres findet sich in der Dokumentation von „stty“. Sind die richtigen Einstellungen für die Zielhardware ermittelt, werden sie in eine der freien Einstellungen des ProgEdi übernommen und sind dann auf Mausklick erreichbar.

Gleiches gilt für den eigentlichen Befehl zum Download. Er wird in eine der freien Einstellungen, z.B Senden, übernommen und ist dann auch per Mausklick erreichbar. Der Konsolen-Befehl nennt sich „cat“ und ist geeignet eine Datei über die serielle Schnittstelle zu senden. Im Beispiel wird die Intel-Hex Datei IS52XTLL.ihx aus dem Pfad „/home/tom/LDaten/“ zum Gerät „/dev/ttyUSB0“ gesendet.

```
cat /home/tom/LDaten/IS52XTLL.ihx > /dev/ttyUSB0
```

6.2 Libusb Geräte

Auch hier ist es wichtig das richtige Gerät zu ermitteln. Dazu das Gerät mit einem freien USB-Anschluss des PC verbinden und in der Konsole den Befehl „lsusb“ eingeben (der erste Buchstabe ist ein kleines L, kein großes i).

Ermittelt durch "lsusb":

```
        tom@tom-TM6595T:~/LDaten/QtProgs$ lsusb
        Bus 002 Device 003: ID 1c7a:0603 LighTuning Technology Inc.
->      Bus 002 Device 004: ID 16d0:0418 MCS
        Bus 002 Device 002: ID 8087:0024 Intel Corp. Integrated Rate Matching Hub
```

Mein Programmiergerät (ID 16d0:0418 MCS) ist hier das zweite Gerät: Bus 002 Device 004.
Nun die Rechte der Geräte prüfen. Hier prüfe ich die Geräte am Bus 002:

```
ls -l /dev/bus/usb/002
```

```
tom@tom-TM6595T:~/LDaten$ ls -l /dev/bus/usb/002
        crw-rw-r-- 1 root root 189, 128 Mai 11 09:04 001
        crw-rw-r-- 1 root root 189, 129 Mai 11 09:04 002
        crw-rw-r-- 1 root root 189, 130 Mai 11 09:04 003
->      crw-rw---- 1 root root 189, 131 Mai 11 13:49 004
```

Mein Gerät (Nummer 004) hat Lese/Schreib-Rechte (rw) für den Besitzer und die Gruppe, gehört aber nur zu Gruppe root und erlaubt mir deshalb als Benutzer keinen Zugriff.

6.2.1 Zugriff vorübergehend einrichten

Zum testen des Gerätes und prüfen der Einstellungen können diese vorübergehend verändert werden. Die Änderungen sind sofort wirksam, aber spätestens beim nächsten Systemstart sind sie wieder zurückgestellt.

Da es sich um ein USB-Gerät handelt, ist es in der Gruppe „plugdev“ (Steckbare Geräte) gut aufgehoben. Ändern der Gruppe für das Gerät:

```
sudo chown root:plugdev /dev/bus/usb/002/004
```

Überprüfen der Änderung:

```
ls -l /dev/bus/usb/002
```

```
        crw-rw-r-- 1 root root    189, 128 Mai 11 09:04 001
        crw-rw-r-- 1 root root    189, 129 Mai 11 09:04 002
        crw-rw-r-- 1 root root    189, 130 Mai 11 09:04 003
->      crw-rw---- 1 root plugdev 189, 132 Mai 11 14:15 004
```

Jetzt befindet sich das Gerät in Gruppe „plugdev“ und ich darf als Mitglied dieser Gruppe auch darauf zugreifen. Sollten dem Gerät die nötigen Schreib/Lese-Rechte fehlen, lassen diese sich mit „chmod“ einstellen. Das „+w“ steht für Schreibzugriff ermöglichen:

```
sudo chmod u+w /dev/bus/usb/002/004
```

Überprüfen der Änderung: ls -l /dev/bus/usb/002

6.2.2 Zugriff dauerhaft einrichten

Um den Zugriff dauerhaft, bei jedem Systemstart freizugeben ist anders vorzugehen als zuvor beschrieben. In einem zugänglichen Verzeichnis (z.B: der Projektordner) eine Textdatei mit dem Extend „.rules“ erstellen, in der die gewünschten Geräte mit ihrer VID, PID, Zugriffsmodus und Gruppe aufgelistet sind:

Im Beispiel hier werden vier Geräte eingerichtet. Ein selbstgebautes Programmiergerät, zwei USB-µC im Bootloadermodus und ein AVR ISP MKII-Programmiergerät:

Datei: usbproger.rules

```
# IS51 libusb-device
ATTRS{idVendor}=="16d0", ATTRS{idProduct}=="0418", MODE="0660", GROUP="dialout"

# AT89C5131 - Bootloader
ATTRS{idVendor}=="03eb", ATTR{idProduct}=="2ffd", MODE="0664", GROUP="plugdev"

# AT89C5122 - Bootloader
ATTRS{idVendor}=="03eb", ATTRS{idProduct}=="2ffe", MODE="0664", GROUP="plugdev"

# AVR ISP MKII
ATTRS{idVendor}=="03eb", ATTRS{idProduct}=="2104", MODE="0664", GROUP="plugdev"
```

Hinweis! -> Hexzahlen in Kleinschreibung, nicht 0x03EB sondern 0x03eb.

idVendor = Vendor-ID (hier 0x16d0, 0x03eb)
idProduct = Produkt-ID (hier 0x0418, 0x2ffd, 0x2ffe, 0x2104)
MODE = Zugriffsmodus (hier 0664: x6xx=owner-rw, xx6x=group-rw, xxx4=other-r)
GROUP = Gruppe für rw(read-write) Zugriff (Der Anwender muß sich darin befinden)

Abschließend ist die Datei, mit Admin-Rechten, in den Ordner "/etc/udev/rules.d" zu kopieren:

```
sudo cp usbproger.rules /etc/udev/rules.d
```

Die Änderung wird dann beim nächsten Neustart von Linux wirksam, oder sofort durch:

```
sudo /etc/init.d/udev stop
sudo udevadm control --reload-rules
sudo /etc/init.d/udev start
```