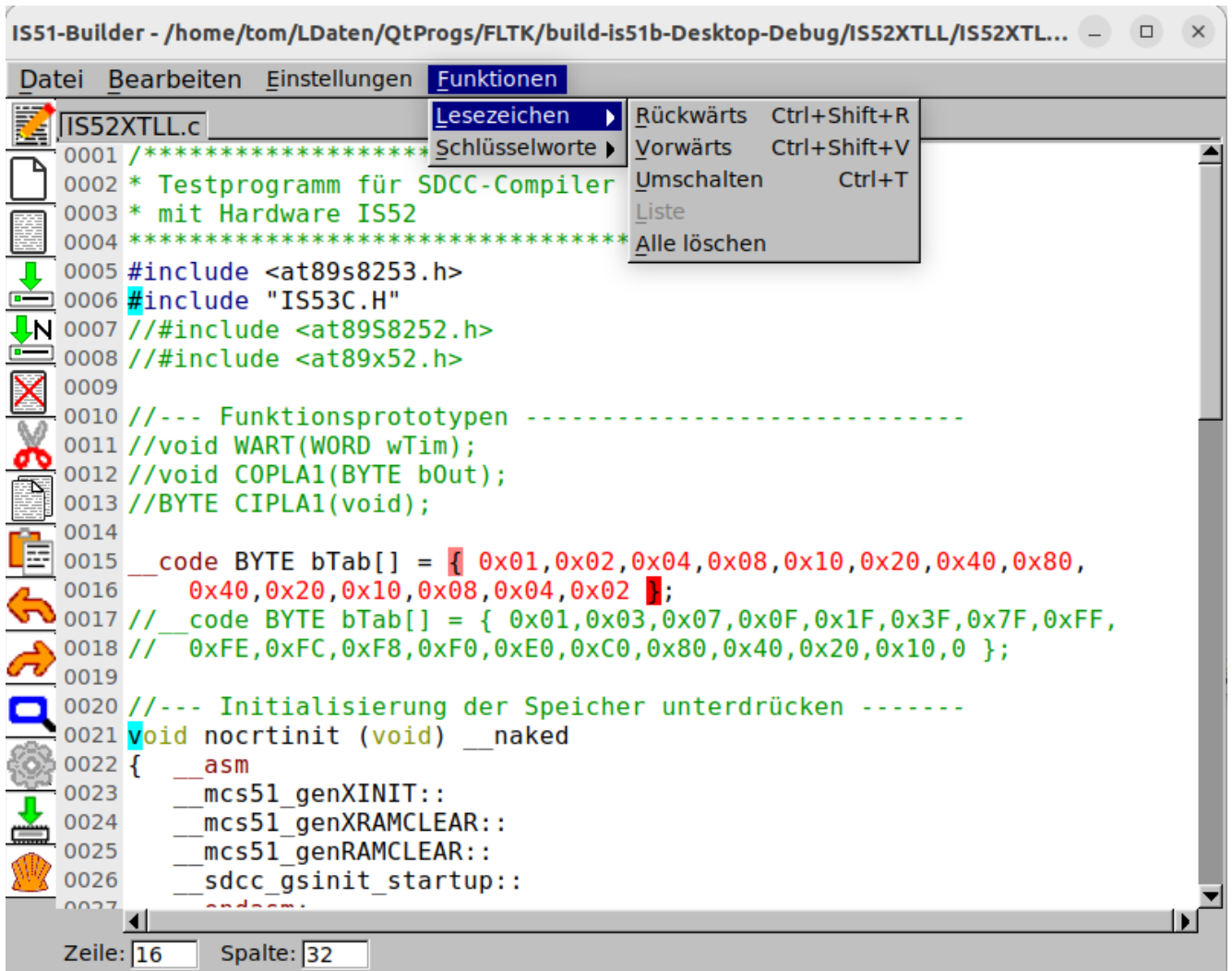


IS51-Builder

Notizen für Anwender



The screenshot shows the IS51-Builder IDE window titled "IS51-Builder - /home/tom/LDaten/QtProgs/FLTK/build-is51b-Desktop-Debug/IS52XTLL/IS52XTLL...". The menu bar includes "Datei", "Bearbeiten", "Einstellungen", and "Funktionen". The "Funktionen" menu is open, showing options: "Rückwärts Ctrl+Shift+R", "Vorwärts Ctrl+Shift+V", "Umschalten Ctrl+T", "Liste", and "Alle löschen". The main editor displays the source file "IS52XTLL.c" with the following code:

```
0001 /*****  
0002 * Testprogramm für SDCC-Compiler  
0003 * mit Hardware IS52  
0004 *****/  
0005 #include <at89s8253.h>  
0006 #include "IS53C.H"  
0007 // #include <at89s8252.h>  
0008 // #include <at89x52.h>  
0009  
0010 //--- Funktionsprototypen -----  
0011 //void WART(WORD wTim);  
0012 //void COPLA1(BYTE bOut);  
0013 //BYTE CIPLA1(void);  
0014  
0015 __code BYTE bTab[] = { 0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,  
0016 0x40,0x20,0x10,0x08,0x04,0x02 };  
0017 //__code BYTE bTab[] = { 0x01,0x03,0x07,0x0F,0x1F,0x3F,0x7F,0xFF,  
0018 // 0xFE,0xFC,0xF8,0xF0,0xE0,0xC0,0x80,0x40,0x20,0x10,0 };  
0019  
0020 //--- Initialisierung der Speicher unterdrücken -----  
0021 void nocrtinit (void) __naked  
0022 { __asm  
0023 __mcs51_genXINIT::  
0024 __mcs51_genXRAMCLEAR::  
0025 __mcs51_genRAMCLEAR::  
0026 __sdcc_gsinit_startup::  
0027 __endasm;
```

The status bar at the bottom shows "Zeile: 16" and "Spalte: 32".

Inhaltsverzeichnis

1	Einleitung.....	4
2	Programm ausführen.....	6
2.1	Einzeldateien.....	6
2.2	Projekte mit mehreren Dateien.....	7
2.3	Download zum Zielsystem.....	8
3	Kurze Übersicht.....	9
3.1	Klammerpaare.....	9
3.2	Lesezeichen.....	9
3.2.1	Nur ein Platzhalter.....	9
4	Beispiele.....	11
4.1	Einfache Textdatei.....	11
4.2	C-Programmdatei.....	11

1 Einleitung

Der IS51-Builder ist eine Integrierte Entwicklungsumgebung (Integrated Development Environment – IDE) für die Mikrocontrollerfamilie MCS51 und speziell in Verbindung mit der Hardware den verschiedenen IS5xx-Platinen und dem SDC-Compiler. Es ist damit aber auch möglich Programme für andere Prozessoren und Mikrocontroller zu schreiben. Der Debugger wird aber nur mit MCS51 arbeiten Da man aus der IDE auch Programme starten kann, lassen sich natürlich auch Debugger anderer Prozessoren/Controller ausführen.

Das Grundkonzept der IDE ist möglichst einfach in der Bedienung zu sein und dem Anwender bei seiner Arbeit möglichst wenig zu behindern. Bei vielen modernen IDE's wird man von gut gemeinten Hilfen förmlich erschlagen und verbringt einen guten Teil der Zeit damit die Folgen der Hilfestellungen wieder zu beseitigen. Meist kann man die Hilfe auch abschalten, aber dann hat man keine Hilfe mehr.

Ich versuche hier Hilfen zur Verfügung zu stellen, sie aber nur nach Aufforderung zu geben. Dadurch vermeide ich, von Hilfe behindert zu werden.

Der Editor mit seinen Funktionen wird öffentlich sein, der Debugger nicht, da er sich auf Hardware bezieht die, ausser mir, niemand hat.

Das Programm ist in C++ und dem FLTK Framework unter und für Linux geschrieben und statisch gelinkt. Dadurch muss der Anwender kein eigenes Rahmenwerk dafür installieren. Das Programm einfach in ein Verzeichnis kopieren und dort starten.

Der Editor ist der aus den FLTK-Beispielen, erweitert um Multifile-, Lesezeichen- und Klammerfindenfunktionen.

Die Werkzeugleiste befindet sich am linken Bildrand, um Platz für mehr Zeilen im Editor zu schaffen.

Durch Rechtsklick im Editorfenster öffnet sich ein lokales Menü, in dem sich später die Hilfsfunktionen finden werden.

Öfter läuft beim Syntxhighlight noch etwas schief, in dem Fall im Menü das highlighting deaktivieren und wieder aktivieren.

2 Programm ausführen

Das Programm wurde statisch gelinkt, dadurch ist alles was es benötigt integriert. Es muß nichts dafür installiert werden. Es muß einfach nur gestartet werden. In der Konsole mit dem Komando: „./is51b“ oder durch anklicken des Symbol in einer grafischen Oberfläche.

Das Zahnradsymbol dient zum übersetzen (compilieren/assemblieren) der momentan aktiven Datei. Der erste Buchstabe des Dateixtend entscheidet über die Art der Übersetzung. Ist der Buchstabe ein „A“ oder „a“ wird die Datei assembliert, bei „C“ oder „c“ wird compiliert. Dazu werden der Compiler/Assembler des SDCC-Paketes verwendet. Es wird nur diese eine Datei übersetzt. Meldungen des Übersetzers werden in der Datei: „dateiname.log“ gespeichert. Fehlermeldungen landen in der Datei: „dateiname.err“. Dabei ist „dateiname“ der Name der aktiven Datei im Editor.



2.1 Einzeldateien

Im Modus für Einzeldateien können die nicht mehr gebrauchten nicht automatisch gelöscht werden. Dazu ist es aber möglich sich ein einfaches makefile mit der benötigten Funktion zu erstellen.

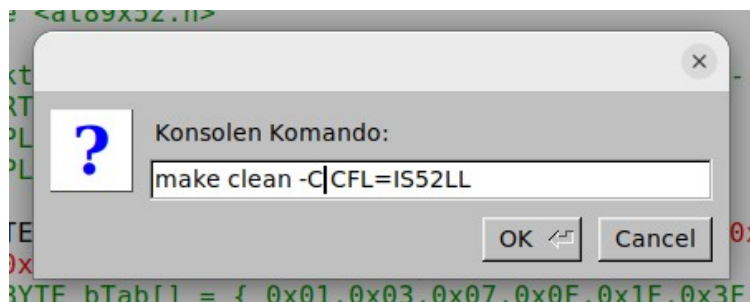
Makefile:

Hinweis: zum löschen der Dateien "make clean CFL=Dateiname ohne Extend" aufrufen.

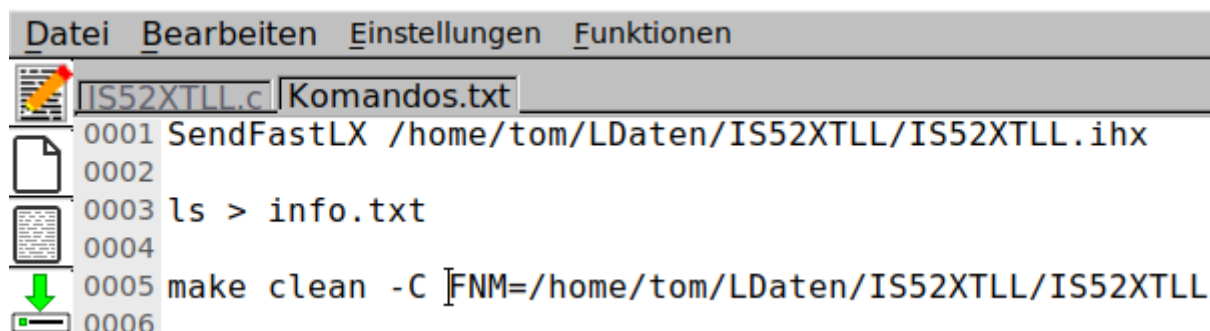
clean:

```
$(RM) $(CFL).adb $(CFL).asm $(CFL).cdb $(CFL).ihx
$(RM) $(CFL).lk $(CFL).lst $(CFL).map $(CFL).mem
$(RM) $(CFL).omf $(CFL).rel $(CFL).rst $(CFL).sym
$(RM) $(CFL).err $(CFL).log
```

Aufgerufen kann „make clean“ über die Shell im Editor werden.



Um das(die) Kommando(s) nicht jedes mal neu eintippen zu müssen, kann man sie in eine eigene Textdatei schreiben und bei Bedarf aus dieser in das Eingabefenster kopieren.



2.2 Projekte mit mehreren Dateien

Um mehrere Dateien zum bilden eines Zielprogramms zusammen zu verarbeiten wird einmalig ein makefile für das Projekt erstellt und kann dann mehrfach benutzt werden. Das makefile wird vorteilhaft im Projektverzeichnis angelegt und wird dann mit „make“ aufgerufen. Braucht man mehrere makefiles mit unterschiedlichen Funktionen im gleichen Verzeichnis, können die makefiles unterschiedliche Namen haben und unter diesen Namen von make bedient werden. Aufgerufen werden sie dann unter der Angabe dieses Namens:

„make -f Dateiname“ – zum Beispiel: „make -f make1“

Innerhalb des makefiles kann nun auch ein Abschnitt „clean“ zum löschen nicht mehr benötigter Dateien sein. Dieser Abschnitt wird dann durch „make clean“ ausgeführt.

Um make beim Aufruf den Namen einer Datei mitzuteilen, kann dieser als Aufrufparameter übergeben werden:

```
make clean -C FNM=/home/tom/Ldaten/IS52XTLL/IS52XTLL
```

übergibt an make die Variable FNM (File NaMe), welche nun innerhalb von make benutzbar ist. Innerhalb wird diese Variable dann als „\$(FNM)“ Verwendung finden und steht für:

```
/home/tom/Ldaten/IS52XTLL/IS52XTLL
```

Im folgenden makefile wird die in \$(FNM) übergebene C-Datei und die zugehörige Assemblerdatei „Startup.asd“ zum Hexfile mit dem übergebenen Namen und dem Extend „.hex“ verarbeitet.

Beispiel:

```
#####
# 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'
#####
# The following lines defines additional directories to search for include files
#INCLUDES := -I"C:\Your Directory\Lab3"

#Here are the sourcefiles used for the project
#CFX = weisnichtwas          hier können auch weitere Quelldateien eingefügt werden.
CFL = $(FNM)
AFL = Startup

#MODEL = --model-large
MODEL = --model-small

# The following line defines flags given to the SDCC C compiler
CFLAGS := -c --debug --verbose $(MODEL) $(INCLUDES)

# The following line defines flags given to the SDCC linker
# Non-specific: --code-loc 0x6000 --xram-loc 0xB000 --model-large
LFLAGS := --debug --verbose --code-loc 0x8100 --code-size 0x8000 --xram-loc 0xC000 --xram-size
0x4000 $(MODEL)

# The following line specifies the default target(s) to build
#all: file.hex
```

```

# The following line specifies the object files that are to be linked together
OBJECTS := $(CFL).rel $(AFL).rel

# The following lines define a rule that sends the object files through the linker to
# create file.ihx which then has to be processed by packihx to create file.hex
file.hex : $(OBJECTS)
    @echo "Linker:" >> $(CFL).log
    sdcc $(LFLAGS) $^ >> $(CFL).log 2>> $(CFL).err
    packihx $(CFL).ihx > $(CFL).hex

# The following rule sends each C file through the preprocessor and creates the asm file
# that is then assembled to create the rel file.
%.rel : %.c
    @echo "Compiler:" > $(CFL).log 2> $(CFL).err
    sdcc $(CFLAGS) $< >> $(CFL).log 2>> $(CFL).err

# The following rule sends each asm file (Not the asm files created by SDCC as an
# intermediate output of the compilation process.) through the assembler to create a rel
# file.
%.rel : %.asd
    @echo "Assembler:" >> $(CFL).log
    sdas8051 -plogff $< >> $(CFL).log 2>> $(CFL).err

# The following rule will clean all the derived objects from your directory. This will
# save you from accidentally typing 'rm *' if you are developing on a UNIX platform.
# Hinweis: zum löschen der Dateien "make clean" aufrufen.
clean:
    $(RM) $(AFL).lst $(AFL).rel $(AFL).rst $(AFL).sym
    $(RM) $(CFL).adb $(CFL).asm $(CFL).cdb $(CFL).ihx
    $(RM) $(CFL).lk $(CFL).lst $(CFL).map $(CFL).mem
    $(RM) $(CFL).omf $(CFL).rel $(CFL).rst $(CFL).sym
    $(RM) $(CFL).err $(CFL).log

```

Fehlermeldungen werden in einer Datei „übergebener Name.err“, Standardmeldungen in „übergebener Name.log“ gespeichert. Ist die Fehlerdatei leer, so hat SDCC keinen Fehler erkannt.

Mit „clean“ werden alle dabei entstandenen Zwischendateien gelöscht. Übrig bleiben nur die Quelldateien und die Hexdatei. „make clean -C Dateiname-ohne-Extend“. Der Aufruf von make über die IDE macht das automatisch, um den korrekten Inhalt des makefile muss man sich selbst kümmern.

2.3 Download zum Zielsystem

Unter Download wird das erzeugte Hexfile über USB zur IS5x übertragen. Da sonst niemand diese Hardware hat ist deren Beschreibung nicht wichtig.



Für abweichende Zielhardware kann das/die entsprechende(n) Kommando(s) in der Shell gegeben werden. Um nicht so viel tippen zu müssen kann man die erforderlichen Kommandos auch in der Kommandodatei ablegen und von dort in die Shell kopieren.

3 Kurze Übersicht

Von den üblichen Tastenkommandos für Editoren sind die meisten umgesetzt. Die zugehörigen Funktionen sind auch über das Menü erreichbar, dort finden sich auch die zugeordneten Tastenkürzel nochmal.

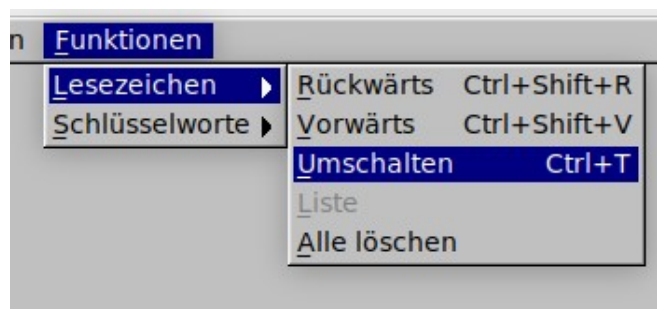
Abkürzung	Beschreibung
FL_COMMAND+'a'	Alles auswählen
FL_COMMAND+'A'	Keine auswählen
FL_COMMAND+'c'	Kopie
FL_COMMAND+'f'	Finden
FL_COMMAND+'g'	Nächstes suchen
FL_COMMAND+'n'	Neu
FL_COMMAND+'o'	Offen
FL_COMMAND+'p'	Drucken
FL_COMMAND+'q'	Beenden (Anwendung)
FL_COMMAND+'r'	Ersetzen
FL_COMMAND+'s'	Speichern
FL_COMMAND+'S'	Speichern unter
FL_COMMAND+'u'	Duplikat
FL_COMMAND+'v'	Paste
FL_COMMAND+'w'	Schließen
FL_COMMAND+'x'	Schneiden

3.1 Klammerpaare

Wird eine Klammer mit der Maus oder der Tastatur markiert, wird die zugehörige Klammer auch rot markiert. Nachdem irgend etwas am Text geändert ist, verschwinden auch die Markierungen. Es werden nur runde- „()“, geschweifte- „{}“ und eckige „[]“ Klammern berücksichtigt, da die spitzen Klammern auch ungepaart im Quelltext vorkommen können.

3.2 Lesezeichen

Im Text können an beliebigen Stellen Lesezeichen gesetzt werden. Durch Umschalten wird ein Lesezeichen gesetzt, oder falls bereits vorhanden, entfernt. Mit Rückwärts und Vorwärts geht es zum jeweils vorigen oder folgenden Lesezeichen. Die Liste um ein Lesezeichen gezielt aufzusuchen ist noch nicht realisiert. Die Funktion „Alle löschen“, löscht alle Lesezeichen unwiderruflich.



Die Arbeitsweise ist dabei modulo, wenn ein Ende erreicht ist, wird am anderen Ende weiter gemacht.

3.2.1 Nur ein Platzhalter

Datei - Neu

Menü: Datei – Neu (strg+n)
Erstellt eine neue Datei mit dem Namen „neu.txt“

Symbol:

**Datei - Öffnen**

Menü: Datei – Öffnen (strg+o)
Öffnet eine bereits bestehende Datei

**Datei - Speichern**

Menü: Datei – Speichern (strg+s)
Speichert eine bearbeitete Datei.

**Datei - Speichern unter**

Menü: Datei – Speichern unter. (strg+shift+s)
Speichert die aktuelle Datei unter einem neuen Namen.

**Datei - Schließen**

Menü: Datei schließen (strg+w)
Schließt die aktuelle Datei.

**Bearbeiten - Ausschneiden**

Menü: Bearbeiten Ausschneiden (strg+x)
Markierten Text ausschneiden und in die Zwischenablage legen.

**Bearbeiten - Kopieren**

Menü: Bearbeiten Kopieren (strg+c)
Markierten Bereich in die Zwischenablage legen.

**Bearbeiten - Einfügen**

Menü: Bearbeiten Einfügen (strg+v)
Inhalt der Zwischenablage einfügen.

**Bearbeiten - Rückgängig (undo)**

Menü: Bearbeiten Rückgängig (strg+z)
Letzte Funktion zurücknehmen.

**Bearbeiten - Wiederherstellen (redo)**

Menü: Bearbeiten Wiederherstellen (strg+y)
Letztes Undo zurücknehmen.

**Bearbeiten - suchen**

Menü: Bearbeiten suchen (strg+f)
Text in Quelldatei finden.

**Bearbeiten - Übersetzen (Compiler/Assembler)**

Menü: Bearbeiten Übersetzen
Quelldatei assemblieren/compilieren.

**Bearbeiten - Download**

Menü: Bearbeiten Download
Hexdatei zum Zielsystem übertragen (nur IS5x).

**Bearbeiten - Shell**

Menü: Bearbeiten Shell
Ausführen von Konsolenprogrammen.



Einstellungen - Syntaxhighlight

Menü: Einstellungen Syntaxhighlight
Umschalten des Syntaxhighlight

Einstellungen - Zeilennummern

Menü: Einstellungen Zeilennummern
Wird noch nicht bedient.

Einstellungen - Intel-Hex

Menü: Einstellungen Intel-Hex
Wird noch nicht bedient. Intel-Hex Editor zum bearbeiten von Hex-Dateien.

Funktionen - Lesezeichen

Menü: Funktionen Lesezeichen ...
Bearbeiten (umschalten, anspringen, löschen) der Lesezeichen.

Funktionen - Schlüsselworte

Menü: Funktionen Schlüsselworte ...
Wird noch nicht bedient. Hilfslisten mit oft benötigten Worten um sie daraus in den Quelltext kopieren zu können. Dies zeigt Mnemonics, C-Befehle, SFR, SFR-Bit, Registernamen und vieles mehr, um sie nicht in verschiedenen Dokumenten nachschlagen zu müssen.

4 Beispiele

Hier finden sich einige Beispiele für den Umgang mit IS51-Builder.

4.1 Einfache Textdatei

Im normalen Textmodus verhält sich der IS51-Builder wie viele andere Editoren auch. Es können Lesezeichen an beliebigen Stellen gesetzt werden. Es werden immer Zeilennummern angezeigt. Die Anzeige der Cursorposition Spalte/Zeile ist unten links in der Statuszeile. Die Funktionen einfügen/überschreiben fehlen noch.

4.2 C-Programmdatei

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

Programm Prog1.c zum kopieren:

```
/*  
* 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
  }
}

```

Programm Prog1.c im Editor:

```

IS51-Builder - /home/tom/LDaten/QtProgs/FLTK/build-is51b-Desktop-Debug/IS52XTLL/Prog1.c
Datei Bearbeiten Einstellungen Funktionen
Prog1.c
0001 /*****
0002 * Testprogramm für C-Compiler *
0003 *****/
0004 #include <8051.h>
0005
0006 //--- Initialisierung der Speicher unterdrücken -----
0007 void noirtinit (void) __naked
0008 { __asm
0009 __mcs51_genXINIT::[
0010 __mcs51_genXRAMCLEAR::
0011 __mcs51_genRAMCLEAR::
0012 __sdcc_gsinit_startup::
0013 __endasm;
0014 }
0015
0016 int main ()
0017 { char cVar;
0018   for(;;)          //Endlosschleife
0019   { cVar = P1;     //Schalterstellung einlesen
0020     P3 = cVar;    //Zu den LED's ausgeben
0021   }
0022 }
0023

```

Compilieren durch Klick auf das Zahnrad der Werkzeugleiste oder Menü: Bearbeiten - Uebersetzen. Dadurch wird eine ganze Reihe Zwischendateien erzeugt, die bei Nichtgebrauch mit Menü: Bearbeiten - Clean wieder entfernt werden können, sofern ein entsprechendes makefile vorhanden ist.

Daraus erzeugte Intel-Hex Datei:

```

:03000000020000FB
:03000600020003F2
:03000300020009EF
:060009008590B080FB228F
:00000001FF

```

Diese HexDatei kann nun zum Zielsystem übertragen und dort ausgeführt werden.