CC166 und MK166

zentrale Tools der Tasking-Toolkette

Gottfried Prandstetter

CC166 ist das zentrale Steuerprogramm, welches die einzelnen Tools der Kette bedient und somit wesentlich zur automatischen Übersetzung beiträgt. Sowohl Benutzer von EDE als auch Entwickler, welche lieber mit einem eigenen Editor und Batch-Dateien arbeiten, werden von CC166 unterstützt. Nachfolgend soll für Benutzer der zweiten Kategorie anhand eines kleinen Beispiels gezeigt werden, wie sich eine komfortable Übersetzung realisieren läßt.

Zentrale Steuerdatei für die Toolkette ist die Datei `makefile`. In dieser Textdatei lassen sich alle Anweisungen aller Übersetzungsstufen unterbringen. Man kann also mit nur einer Steuerdatei ein ganzes Projekt übersetzen.

Diese Übersetzung erfolgt dann intelligent, d. h. nur jene Module, welche tatsächlich verändert oder noch nicht bearbeitet wurden, werden übersetzt.

Gestartet wird dieser Übersetzungsvorgang durch den Aufruf von MK166. Daraufhin sucht und analysiert MK166 `makefile`, entscheidet welche Teile des Projekts zu übersetzen sind und überlässt CC166 die Ausführung der Übersetzungsarbeit. Dazu übergibt er an CC166 eine Reihe von Parametern. Da die Menge an Parametern schnell die zulässige Länge einer DOS-Eingabezeile überschreiten, übergibt er diese Parameter bei Bedarf in einem File. CC166 weiß mit welchen Tools es den Auftrag zu übersetzen hat, ruft diese auf, generiert dazu seinerseits die nötigen Tool-Parameter bzw. reicht manche Parameter durch.

Obwohl man also ein Projekt auch mit CC166 allein weitgehend automatisieren kann, kommt wahrer Luxus erst durch die Zusammenarbeit von MK166 und CC166 auf.

Das `makefile` - Beispiel soll ein kleines Programm für SAB167 als Zielprozessor übersetzen, d. h. einige der Anweisungen dienen dazu von 166 auf 167 umzuschalten (`EXTEND`). Kommentarzeilen beginnen immer mit einem # als Umschaltzeichen, Befehlszeilen werden nur bis zum Auftreten eines # abgearbeitet. Weiters werden die Makros `CFLAGS` und `LDFLAGS` benutzt. Die Statments in `CFLAGS` werden an den Compiler bzw. Assembler durchgereicht, `LDFLAGS` wird für den Linker/Locator verwendet. Zum Auflösen dieser Makros sucht und verwendet MK166 die Datei `MK166.MK` im Verzeichnis `ETC`. In dieser Datei sind noch weitere Makros vordefiniert, für eine normale Übersetzung reichen die beiden genannten.

Enthält das `makefile` Syntax-Fehler, so wird nicht einfach abgebrochen sondern MK166 versucht kleinere Verstöße wie überflüssige oder fehlerhafte Parameter zu „übersehen“. Auch Zeilen, welche keinerlei Sinn ergeben, also Kommentare ohne vorangestelltes # werden verdaut. Trotzdem gibt es z.B. Probleme, wenn hinter dem Backslash, welcher hier benutzt wird um Zeilen zusammenzuhängen, noch Kommentar eingefügt wird.

Was bei der Übersetzung geschieht, kann man bei aktiven –v Switches gut verfolgen. Die dadurch erzeugte Informationsflut kann aber auch störend wirken und sollte, wenn alles zufriedenstellend läuft, ausgeschaltet werden. Eventuelle Fehlermeldungen kann man dann besser erkennen.

Auf der nächsten Seite nun das komplette `makefile`. Es ist praxiserprobt und sollte auch in Ihren Projekten arbeiten. Stellen Sie die File-Namen und Parameter entsprechend Ihren Anforderungen ein. Selbstverständlich muss für jedes Projekt ein eigenes Directory angelegt werden und ein Suchpfad auf das BIN-Verzeichnis existieren.

Was geschieht nun wenn man dieses Makefile übersetzen läßt? Das Kommando dazu lautet schlicht: `MK166` Wir gehen davon aus, dass alle Files neu übersetzt werden müssen. MK166 generiert folgende Aufrufe: Dreimal wird CC166 aufgerufen, um *.obj – Files zu erzeugen

```
cc166 -c -v -tmp -s PRINT -S -O2 -x EXTEND t.c
cc166 -c -v -tmp -s PRINT -S -O2 -x EXTEND serio.c
cc166 DEF(MODEL,SMALL) DEF(C167) DEF(EVA,0) DEF(MUXBUS,1)
start.asm –c-v
```

Ein Parameter-File wird erzeugt:

```
mk01674a.tmp:
LISTSYMBOLS
'me rom  (0000h TO 07fffh)  me rom (18000h TO 27fffh)
 me ram  (0e000h TO 0e7ffh) me ram (30000h TO 3ffffh)
 cl(CFAR (30000h TO 30fffh))'
'me ir(0f000h)  re me(0f200h TO 0f5ffh) re pp(PECC0)'
```

```
# Abhängigkeiten der Files werden definiert:
# t.hex wird erzeugt, abhängig von t.obj serio.obj start.obj
all             : t.hex
t.hex           : t.obj serio.obj start.obj
# Anzeige d. Tool-Kommandos und Listfiles erzeugen:
VERBOSE = -v -tmp -s PRINT
#            -tmp löscht tmp.-Dateien nicht
#                 -s mergt C-Code in *.src und *.lst
#                    PRINT  hebt default NOPRINT auf,erzeugt *.lst-Files
# Switches f. C-Compiler:
CFLAGS X        = -S -O2
#                 -S Static allocation of automatics
#                     O1 = default, O2 = size O3 = speed optimize
# Das Makro CFLAGS wird zusammengesetzt aus ....
CFLAGS    = $(VERBOSE) $(CFLAGS X) -x EXTEND  #-t
#                                              -t : Modul-Daten anzeigen
# Switches f. Linker:Speicherbereiche deklarieren:
LDFLAGS_X = "'me rom  (00000h TO 07fffh) \
             me rom  (18000h TO 27fffh) \
             me ram  (0e000h TO 0e7ffh) \
             me ram  (30000h TO 3ffffh) \
             cl(CFAR (30000h TO 30fffh))'"
# Zuordnung physischer-logischer Speicher über Variablenklasse CFAR
# Internes RAM, internes XRAM, PEC-Pointer reservieren:
SET EXT LOC = "'me ir(0f000h)\
               re me(0f200h TO 0f5ffh) \
               re pp(PECC0)'"
# Makro LDFLAGS zusammensetzen:
LDFLAGS     = LISTSYMBOLS $(LDFLAGS X) $(SET EXT LOC) $(SET MODEL)-v -x
# LISTSYMBOLS: Tabelle erzeugen
# switch -x setzt richtige Library über CC166!!!
# switch -v zeigt Tool-Kommandozeile
# switch -ihex serzeugt Hex-File
# start.obj soll aus start.asm erzeugt werden:
start.obj : start.asm
# Sonderfall CSTART.ASM !!!
# An M166 sollen DEFINES zur bedingten Übersetung übergeben werden.
# Hier wird CC166 direkt mit jenen Parametern aufgerufen, welche an
# M166 übergeben werden sollen.
# Achtung! Die folgende Befehlszeile muss eingerückt werden !!!
   cc166 DEF(MODEL,SMALL) DEF(C167) DEF(EVA,0) DEF(MUXBUS,1) start.asm -c -v
# -c : nicht linken, nur übersetzen mit M166 und A166
# -v : zeige Aufrufzeilen detailliert
# Radikal alle Output-Files löschen mit: MK166 clean
clean:
        del *.obj
        del *.abs
        del *.src
        del *.map
        del *.lno
        del *.h86
        del *.hex
        del *.lst
```

```
-v -x
t.obj serio.obj start.obj
```

CC166 wird mit dem Parameter-File gefüttert, um das Projekt zu linken und ein HEX-File zu erzeugen:

```
cc166 -cf -ihex -o t.hex -f mk01674a.tmp
```

Hier ist das Projekt fertig übersetzt und das Ergebnis `t.hex` liegt vor. Viermal wurde CC166 gerufen, um die Übersetzungstools zu starten. CC166 generiert seinerseits die Parameter für die Tools, welche er zur Übersetzung braucht.

In der ersten Zeile hat CC166 folgendes getan:

```
 c166 t.c -o t.src -e -s -S -O2 -x
 a166 t.src TO t.obj NOPR EXTEND PR
```

In der letzten Zeile (4. Aufruf) hat er das Projekt gelinkt und das HEX-File erzeugt:

```
l166 LOC TO cc09531c.tmp
PTOG t.obj serio.obj start.obj ext\c166s.lib ext\f166
s.lib PR(t) LISTSYMBOLS
ME rom  (0000h TO 07fffh)  me rom  (18000h TO 27fffh)
me ram  (0e000h TO 0e7ffh) me ram  (30000h TO 3ffffh)
cl(CFAR (30000h TO 30fffh))
ME ir(0f000h)  re me(0f200h TO 0f5ffh) re pp(PECC0)
ihex166 cc09531c.tmp t.hex
```

Die Möglichkeiten von Make-Files sind ähnlich flexibel wie die von Batch-Dateien. Hier sollte nur ein brauchbares, einigermaßen übersichtliches Minimal-Gerüst beschrieben werden. Erweiterungsmöglichkeiten entnehmen Sie bitte den Dokumentationen der jeweiligen Programme.