3 Der Debugger

3.1 Überblick

Die Funktion des Debuggers beruht darauf, dass an das Ende des zu debuggenden Programms ein Monitor mit einer Länge von knapp 1 kByte angehängt wird. Dieser Monitor wird nach dem Reset des Kontrollers aufgerufen und die Interruptvektoren für den NMI (NMITRAP), für den Empfangsinterrupt der ser. Schnittstelle ASC0 (S0RINT) und den Interrupt für undefinierten Opcode (BTRAP) werden vom Compiler auf den Anfang dieses Monitors umgelegt. Da der Monitor hinter die eigentlichr Anwendung geladen wird, ist sichergestellt, dass diese mit oder ohne Monitor im selben Speicherbereich ausgeführt wird.

Der Monitor kann dann die eigentliche Anwendung starten, Haltepunkte setzen oder löschen und ähnliche Aktionen ausführen. Das PC–Programm steuert diesen Monitor über wenige, sehr primitive Anweisungen, wobei die Kommunikation zwischen diesen beiden bisher über die ser. Schnittstelle erfolgt.

Die PC–Anwendung wurde mit Lazarus erstellt und sollte damit auf allen von dieser Umgebung unterstützten Plattformen übersetzbar sein, wenn die jeweilige Debuggschnittstelle darunter verwendbar ist. Im Moment ist diese sicher nur als Machbarkeitsstudie zu betrachten, die ich nach und nach erweitern werde. Es macht aber schon Spaß, damit einer Anwendung unter die Haube zu schauen.

Der Monitor wurde natürlich in Oberon geschrieben, wobei relativ viel Assembler in der Verbatim–Umgebung benötigt wurde.

3.2 Einschränkungen

Da der C167 und andere Kontroller dieser 2. Generation der Familie über keine echte Debug–Schnittstelle verfügen, muss zum Debuggen deren serielle Schnittstelle ASC0 verwendet werden. Dies bringt zwei Einschränkungen mit sich:

1. Diese ser. Schnittstelle kann in der zu debuggenden Anwendung nicht verwendet werden. Nachrichten, die über sie gesendet werden, laufen ins Leere und die zugehörigen Antworten können nicht übermittelt werden. Auch eine Initialisierung dieser

Schnittstelle kann das Debuggen erschweren, wird z.B. die Baudrate darin geändert, versteht der Kontroller das Break des Debuggers nicht mehr und kann daher nicht darauf reagieren. Dann hilft nur noch ein NMI zum Aufruf des Monitors.

Konsequenz: Alle Zugriffe auf diese ser. Schnittstelle müssen auskommentiert werden, ebenso die Initialisierung dieser Schnittstelle, sofern diese nicht ebenso konfiguriert wird, wie im Debugger (38400 8N1).

Modernere Mitglieder dieser Kontrollerfamilie verfügen über eine zusätzliche serielle Schnittstelle ASC1 und diese kann natürlich uneingeschränkt genutzt werden, z.B. mit einem Terminalprogramm.

2. Um ein Programm mittels Haltepunkten unterbrechen zu können, muss der Debugger "undefinierte Instruktionen" in das Programm einfügen. Dies funktioniert natürlich nur, wenn das Programm in RAM liegt, einzelne Zellen des Flash–Programmspeichers können nicht überschrieben werden.

Wer angesichts dieser Einschränkung die Nase rümpft sollte bedenken, dass auch die Profis von Keil und Tasking mit diesem Problem konfrontiert waren und keine bessere Lösung gefunden haben. Diese wird erst mit den Nachfolgern der Familie, den XC16x und XE16x möglich, die über eine JTAG–kompatible bzw. über eine 2–Draht–Schnittstelle zum Debuggen verfügen. Die Kommunikation des Monitors mit dem Debugger wird auf dem PC über eine Zwischenschicht realisiert, die zukünftige Verbesserungen in dieser Beziehung zulässt.

3.3 Vorbereitungen

Der Debugger benötigt eine Reihe an Dateien, die alle im selben Verzeichnis liegen müssen, damit File–Select–Orgien vermieden werden. Dies sind:

Bootstraploader: Dieser lädt die Anwendung in drei Schritten in den Programmspeicher. Die erste Stufe des Laders, ein exakt 32 Byte langes Assemblerprogramm, musste ich seit meinen ersten Versuchen nicht mehr ändern und habe sie daher als String fest in den Debugger übernommen.

Die zweite Stufe des Laders muss die Buskonfiguration vornehmen und deshalb auf die jeweilige Hardware angepasst werden. Ein Beispiel mit Buildskript lege ich den Compilerquellen bei. Diese wird als **stage2.bin** kompiliert und assembliert, wobei auf die nötigen Compileroptionen zu achten ist (s. Buildskript).

Monitor: Das File monitor.mod wird mit der Option `-im1` kompiliert. Dabei entsteht ein Assemblerfile **monitor.asm** ohne Startupcode, Interruptvektoren und solche Sachen. Zusätzlich wird durch die Option eine Symboldatei dbgmonitor.sym erzeugt, die hier jedoch nicht benötigt wird und gelöscht werden kann.

Anwendung: Die hat natürlich ihren eigenen Namen, ich nenne sie einfach anwendung. Sie muss mit der Compileroption `-db1` kompiliert werden, wodurch der Monitor eingebunden wird. Zusätzlich werden die Zeilennummern des Quelltextes und die Startadresse der Anwendung als Kommentare in die erzeugte Assemblerdatei geschrieben. Bei der Assemblierung erstellt der Assembler ein Listing, das dann diese Zeilennummern und die zugehörigen Codeadressen beinhaltet. Nach der Assemblierung muss die erzeugte Ausgabedatei in das Intel–Hex–Format umgewandelt werden, damit der Bootstraploader sie übertragen kann. Die für den ganzen Vorgang nötigen Schritte und alle Optionen können den Buildskripten meiner Beispiele entnommen werden. In verkürzter Version:

```
obcc -db1 anwendung
asl -L -E -x anwendung.asm
p2hex -F Intel -r \$-\$ anwendung.p
```

Unter Windows heißt der Assembler asw statt asl und beim p2hex müssen die Backslashes vor den Dollarzeichen weggelassen werden. Sonst geht das genauso.

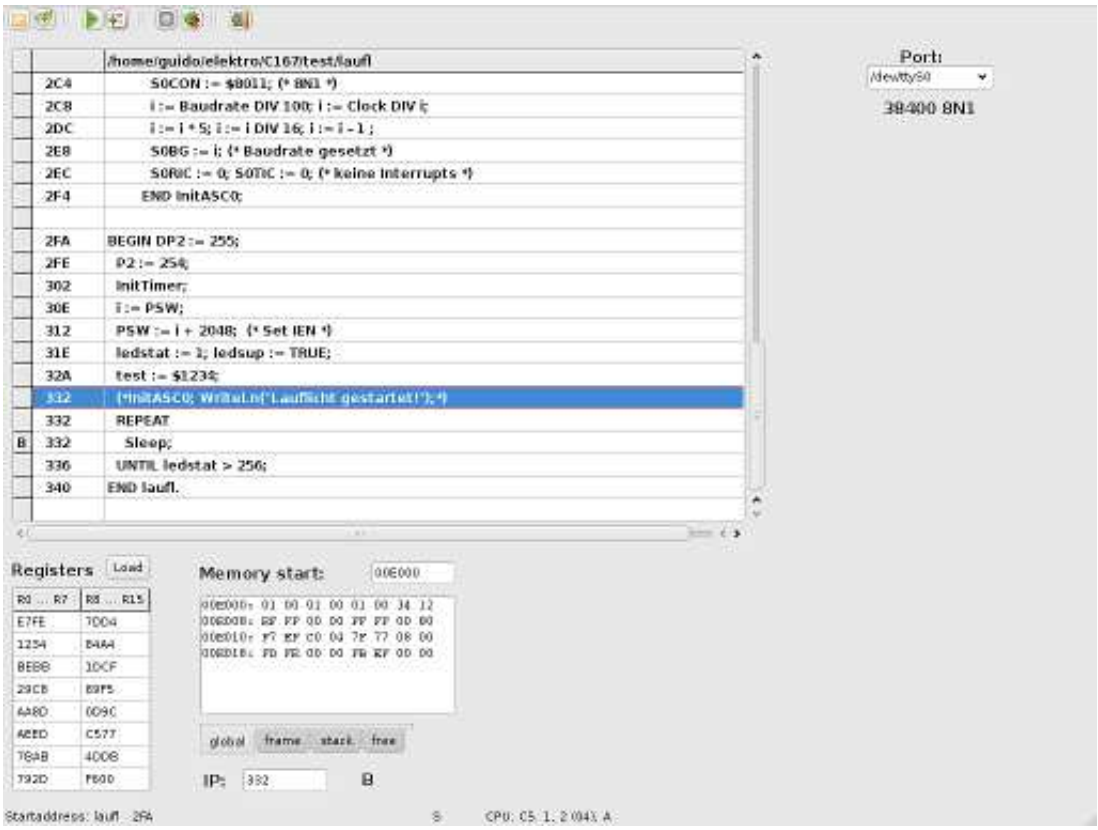
Letztendlich resultieren drei Dateien, die vom Debugger benötigt werden: **anwendung.mod**, **anwendung.hex** und **anwendung.lst**.

3.4 Der Debugger obcdbg

Die Steuerung des Debuggers erfolgt über die sieben Speedbuttons oben, die eigentlich selbsterklärend sind. Verweilt man kurz mit dem Mauszeiger über diesen, wird ein "Hint" zur Bedeutung angezeigt. Von links nach rechts haben wir:

1. open: Öffnet ein Projekt.
2. connect to target/disconnect: Stellt die Verbindung zum Target her bzw. trennt diese.
3. run: Startet die Anwendung auf dem Target.
4. step: Führt einen Einzelschritt der Anwendung an der aktuellen Adresse aus.
5. break: Unterbricht die Anwendung auf dem Target, es wird in den Monitor zurückgekehrt.
6. reset application: Neustart der Anwendung und des Monitors.
7. exit debugger: habe ich vergessen, was war das nochmal?

Ich habe mich bemüht, immer nur die sinvollen Buttons zu aktivieren, so dass eine Fehlbedienung erschwert wird, dies ist aber sicher noch nicht perfekt.



Mit dem Öffnen wird eine Fileselectbox aufgerufen, in der man das zu debuggende Projekt in Form der .mod–Datei auswählt. Dieser Quelltext wird in das Stringgitter geladen und aus dem zugehörigen Listing die Speicheradressen der einzelnen Zeilen extrahiert und mitangezeigt. Auch die Startadresse wird aus dem Listing gelesen, diese unten als IP angezeigt und im Stringgitter an die Quelltextposition dieser Adresse gesprungen. Der Scrollbalken springt hierbei leider nicht mit, da bin ich mal auf die Windowsversion gespannt. Das Stringgitter ist in diesem Moment für den Nutzer noch gesperrt.

Hat man rechts oben das Device eingetragen, über das die Verbindung zum Target hergestellt wird, kann man “verbinden” anklicken. Hierbei wird der Bootstraploader angesprochen und nacheinander die 3 Dateien zum Target übertragen. Im Statusbalken sieht man momentan rechts noch den Erfolg. Anschließend ist das Stringgitter freigegeben, man kann darin herumscrollen und mit einem Doppelklick auf eine Zeile einen Haltepunkt setzen bzw. genauso wieder löschen. Die Haltepunkte werden durch ein “B” in der linken Spalte angezeigt.

Die Anwendung kann dann gestartet, gestoppt oder in Einzelschritten durchlaufen werden. Im gestoppten Zustand kann man sich Programmdaten anschauen wie links unten den Inhalt der Register oder daneben Speicherinhalte. Durch die von–Neumann-Architektur der C16x kann damit wirklich alles besichtigt werden. 4 verschiedene Speicher-

bereiche können mit den Reitern angewählt werden, im Editierfeld über der Anzeige kann die Anfangsadresse des gewünschten Bereichs eingegeben werden. Hat man diese geändert, bringt ein Klick auf “Load” die neuen Werte zur Anzeige, sonst werden bei jedem Halt des Targets die gewählten Werte aktualisiert.

Wenn man sich mal völlig verirrt hat, kann das gestoppte Programm zurückgesetzt und die Session erneut gestartet werden.