Technische Hochschule Leipzig
Fachbereich Automatisierungstechnik

Diplomarbeit

Thema:

Integrierte Test- und Entwicklungsumgebung für den Mikrocontroller MC68HC11

vorgelegt von: Oliver Thamm, geb. 20. 1. 1968

Betreuer: Prof. Dr.-Ing. M. Sturm

verantwortlicher Hochschullehrer: Prof. Dr.-Ing. habil. H. G. Woschni

Leipzig, am 28. 2. 1994

Thesen zur Diplomarbeit

- Der überwiegende Teil der für die Automatisierungstechnik typischen Mikrocontroller-Algorithmen lassen sich auch ohne Einsatz komplizierter und aufwendiger Hardware Debugging Tools, wie z.B. In-Circuit-Emulatoren, austesten und in Betrieb nehmen.

- Eine Einzelschrittabarbeitung von MC68HC11-Software läßt sich, auch ohne Verwendung zusätzlicher externer Debugging-Hardware, alleine mit den Ressourcen der MCU durchführen.

- Für die Realisierung einer rein softwarebasierten Einzelschrittabarbeitung genügt bereits die Belegung eines Teilbereich des internen 256 Byte-RAMs des MC68HC11 sowie die Benutzung der asynchronen seriellen Schnittstelle des Mikrocontrollers.

- Durch Verlagerung komplexer Funktionen auf einen Personalcomputer läßt sich, selbst bei den üblicherweise knappen On-Chip Speicher-Ressourcen eines Mikrocontrollers, ein komfortables Entwicklungssystem mit eingebetteten Debugging Tools implementieren.

- Unter Nutzung der vorhandenen Firmware Features des MC68HC11 ist das Beschreiben des internen RAM der MCU ebenso ohne Bedienhandlung am Mikrocontroller-Zielsystem möglich, wie auch die Programmierung des internen EEPROM und das Starten von Software auf dem Zielsystem im Realtime- und im Trace-Mode.

- Die Zusammenfassung verschiedener Utility-Programme zu einer integrierten Software erleichtert deren Handhabung. Dabei erhöht die Verwendung bereits verbreiteter, bekannter Benutzeroberflächen die Akzeptanzbereitschaft des Anwenders und reduziert die erforderliche Einarbeitungszeit.

- Durch die Bereitstellung einer weitgehend intuitiv handhabbaren Entwicklungsumgebung, unterstützt durch eine Bibliothek mit vorbereiteten Low Level Routinen, ist es - insbesondere im Praktikumsbetrieb - möglich, auch komplexere Aufgabenstellungen in kurzer Zeit zu realisieren.

Inhaltsverzeichnis

1. Vorbemerkungen

Die vorliegende Arbeit wurde realisiert unter Verwendung der Produkte folgender Firmen:

- Turbo C++ 3.1, Borland GmbH
- Turbo Vision Library, Borland GmbH
- ASMHC11, Motorola Ltd. (Freeware)
- ZWERG11A Einplatinenrechner, MCT Paul & Scherer GmbH, Berlin
- Ami Pro 3.0, Lotus Development Corporation (Dokumentation)

Der Dank des Autors gilt all jenen, die durch Anregungen und Gespräche zum Gelingen der Hard- und Software sowie dieser schriftlichen Arbeit beigetragen haben, insbesondere und an erster Stelle jedoch Herrn Prof. M. Sturm in seiner Eigenschaft als Betreuer und Partner.

2. Einleitung

Ausgangspunkt für die vorliegende Arbeit war die Notwendigkeit der Schaffung eines Praktikum-Labors zur Unterstützung der Ausbildung im Fach Mikrorechentechnik an der HTWK Leipzig (FH). Die inhaltliche Neuorientierung der Lehrveranstaltung sollte wirksam durch Einrichtung mehrerer PC-basierter Arbeitsplätze unterstützt werden. Als Mikrocontroller wurde aufgrund des modernen Designs und der weiten Verbreitung die Motorola MCU (Micro Controller Unit) MC68HC11 ausgewählt. Diese MCU läßt sich quasi ohne externe Hardware in Betrieb nehmen, selbst Programm- und Datenspeicher sind integriert und in einfacher Art und Weise für den Benutzer zugänglich.

Besonders für Ausbildungszwecke ist es notwendig, erstellte Programme nicht nur auf das Zielsystem zu übertragen, und dann die Reaktion der MCU zu beobachten, sondern auch komfortable Testmöglichkeiten nutzen zu können. Fehlender Einblick in das Zielsystem verzögert nicht nur die Implementierung einer Software, die Behandlung als "Black Box" würde es auch vereiteln, daß der Benutzer tatsächlich die Abläufe und Zusammenhänge innerhalb der MCU durchschauen lernt.

Zu Beginn des Kontaktes mit einem Mikrocontrollersystem weiß der Benutzer i.d.R. noch nicht, wesentliche von unwesentlichen Faktoren zu trennen. Äußerst hilfreich ist in diesem Stadium deshalb die Möglichkeit, ein Programm schrittweise abzuarbeiten. Der Single-Step-Betrieb, oder auch Trace-Mode, ist eine der wichtigsten Debugging-Features. Softwaresimulatoren, wie z.B. das Programm TESTE68 [5], ermöglichen zwar die Simulation der Vorgänge innerhalb einer MCU, es fehlt jedoch der entscheidende Kontakt zur "Außenwelt". Ein Anwender fühlt sich ungleich mehr bestätigt in seinem Tun wenn eine Leuchtdiode auf der Mikrocontrollerplatine blinkt, als wenn diese Funktion durch ein von 0 auf 1 wechselndes Zeichen auf dem Bildschirm des Simulators nachvollzogen wird.

An heutige Softwarelösungen werden (zu Recht) höhere Ansprüche bezüglich der Gestaltung des Benutzerinterface und der erzielten Effizienz gestellt, als dies noch in den Anfangszeiten des Personalcomputers der Fall war. Im Bereich der hardwarenahen Tools, und insbesondere im Bereich der Cross-Software für Mikrocontroller, ist - dessenungeachtet - weiterhin ein Hang zum maximalen Verzicht auf komfortable Bedienung zu verzeichnen. Eine (mehr oder minder) kryptische Syntax eines über Kommandozeileneingaben zu steuernden Programms ist weder State-Of-The-Art, noch sehr ermutigend für Neulinge aus dem Kreis der Anwender. Anliegen der vorliegenden Arbeit war es daher, auch auf diesem Gebiet eine akzeptable Lösung zu erarbeiten.

Dieses waren die Prämissen. Es sollte letztendlich eine Entwicklungs-Software für die Mikrocontroller der ´HC11-Serie entstehen, welche sowohl alle notwendigen Bestandteile zum Schreiben und Übersetzen eines Sourcecodes, die Möglichkeit des Download zum Zielsystem, einen Single-Step-Mode für Testzwecke und alle notwendigen Werkzeuge und Hilfsmittel integriert in eine SAA/CUA-angelehnte, komfortable Benutzeroberfläche beinhaltet. Der Hardwareaufwand sollte minimal gehalten werden, und ebenso die entstehenden Kosten. Das fertige Produkt sollte sowohl für die Ausbildung als auch für umfangreichere Softwareprojekte tauglich sein.

Neue Wege waren zu gehen, um ungeachtet der äußerst "schlanken" Hardware einen Einzelschrittbetrieb zu realisieren. Die dafür erarbeitete Lösung ist als innovative Neuheit besonders hervorzuheben und bildet daher auch den Kern der vorliegenden Ausarbeitung.

3. Testmöglichkeiten für (8-Bit-) Mikrocontroller

Für die Inbetriebnahme von Mikrocomputer- bzw. Mikrocontroller-Systemen haben sich eine Vielzahl Verfahren etabliert, über die im folgenden Abschnitt ein Überblick gegeben werden soll. Die Betrachtungen sollen dabei auf die hier zur Diskussion stehenden 8-Bit MCU´s eingeschränkt werden.

3.1. PC-gestützte Software Simulation

Eine softwaremäßige Simulation der Vorgänge in der MCU läßt sich ohne Zuhilfenahme einer realen Zielhardware vornehmen. Ein PC o.ä. dient zur Darstellung aller Register und des Speicheradreßraumes der MCU. Portein- und -ausgaben können, wie im Beispiel der Simulationssoftware TESTE68 [5], ebenfalls vorgegeben bzw. beobachtet werden. Ein Echtzeittest ist dagegen nicht möglich. Die Analyse des Verhaltens peripherer Baugruppen ist nicht oder nur hypothetisch möglich. Das Interruptverhalten ist nicht testbar. Und schließlich ist eine reine Simulation aus didaktischen Gründen nicht günstig, es fehlt der direkte Kontakt zur untersuchten Hardware.

3.2. Trial & Error Methode

Die einfachste (und leider auch unkomfortabelste) Methode, ein Mikrocontroller-Programm zu testen, ist das Programmieren eines EPROM. Dieser EPROM wird in einen Speichersockel des Zielsystems gesteckt. Nach Anlegen der Betriebsspannung startet das Programm. Treten Fehlfunktionen auf, muß meist ein Blick in die Quelle genügen , um den Fehler aufzudecken. Nachteile: Das Löschen von EPROMs dauert etliche Minuten, Testroutinen irgendeiner Form müssen explizit in der Software eingebaut werden, die Benutzerkommunikation bewegt sich meist auf unterster Stufe.

3.3. EPROM Simulator

Während die vorstehende Methode kaum ernsthaft empfohlen werden kann, stellt der Ersatz des EPROMs durch einen EPROM-Simulator einen wesentlichen Vorteil dar. Ein EPROM-Simulator wird meist über die parallele Schnittstelle eines PC geladen, der zeitraubende Programmiervorgang entfällt. Für das Laden eines 32 KB Speicherbausteins benötigt ein EPROM-Simulator typischerweise 2 bis 5 Sekunden. Die mechanische Belastung des Zielsystems durch wiederholtes Stecken und Entfernen eines IC´s entfällt ebenso. Durch die kurzen Turn-Around Zeiten ist es möglich, verschiedene kleine Modifikationen an der Software ohne nennenswerten Zeitverzug auszutesten. Die restlichen Nachteile des oben genannten Verfahrens bleiben jedoch erhalten. Das folgende Bild zeigt schematisch und beispielhaft die Verbindung zwischen Zielsystem, EPROM-Simulator und PC:

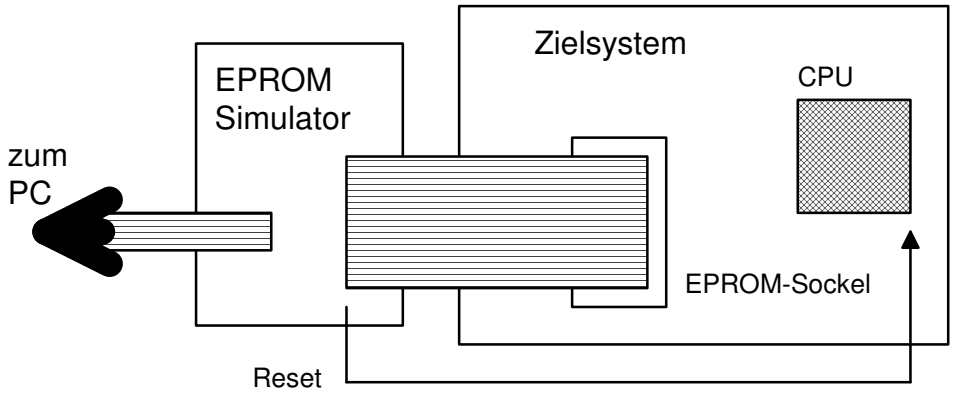

Bild 1: Anschaltung eines EPROM-Simulators

3.4. Logic-Analyzer

Vorausgesetzt, es existiert ein externes Bussystem, so kann man die anliegenden Adreß- und Datensignale auch elektrisch messen bzw. aufzeichnen. Dazu eignet sich ein Logic-Analyzer. Die gewonnenen Aufzeichnungen sind jedoch sehr abstrakt und lassen nur sehr erfahrene Assemblerprogrammierer einen schnellen Rückschluß auf die vorgefundenen Sachverhalte ziehen. Es sind nur elektrische Signale zugänglich, es existieren keine weiteren Informationen über den Sinngehalt der gemessenen Signale (Befehle, Daten...). Hinzu kommt der oft recht hohe Preis für derlei Meßgeräte, die eine Anwendung hierfür i.d.R. nicht sinnvoll erscheinen lassen.

3.5. Hardware Emulator

Die umfangreichsten Test- und Diagnosemöglichkeiten eröffnet der Einsatz eines Hardware-Emulators. Ein solches Gerät wird typischerweise mittels eines Pods (Adapters) statt der MCU auf die Zielhardware aufgesteckt und erlaubt so den Zugriff auf sämtliche externen Signale. Zugriff auf die Register und interne Peripherie ist i.d.R. uneingeschränkt möglich. Der Anwender hat dadurch das breiteste Repertoire an Hilfsmitteln zur Programmentwicklung aller hier vorgestellten Lösungsansätze zur Verfügung. Ebenso hoch wie der Grad der Universalität sind aber auch die Anschaffungskosten. Für den konkret angestrebten Zweck (Schaffung mehrere Ausbildungsplätze) handelt es sich also keineswegs um die ideale Lösung. Für nichtkommerzielle Anwender und kleinere Entwicklungsabteilungen bzw. -firmen dürfte die Anschaffung eines Hardware Emulators auch oft an Kostengründen scheitern. Unterstützt wird die Entscheidung gegen ein solches Hilfsmittel dadurch, daß die zu einem Hardware Emulator mitgelieferte Bediensoftware aufgrund der vielfältig gebotenen Möglichkeiten auch eine nicht zu vernachlässignede Einarbeitungszeit erfordert.

3.6. "mitdenkende" Softwarelösung

Es ist allgemeine Praxis, einen Unterbrechungspunkt im Anwenderprogramm dadurch zu plazieren, daß man an die betreffende Stelle einen anderen Befehlscode einarbeitet, den Code also (temporär) modifiziert. Der eingesetzte Befehl ist entweder ein Return-Befehl (Unterprogramm-Ende) oder ein Befehl zur Erzeugung einer Exception bzw. eines Software-Interrupt (Prozessor-spezifisch). Für eine praktikable Handhabung dieser Methode ist es einerseits notwendig, den Programmspeicher leicht ändern zu können (RAM oder EEPROM oder ggf. Flash-Speicher), andererseits muß man auch in Kenntnis der genauen Lage des zu realisierenden Breakpoints sein.

Handelt es sich um einen normalen Unterbrechnungspunkt im laufenden Programm, ist es sehr wohl bekannt, wo diese Breakpoint-Adresse liegt, schließlich ist es der Wunsch des Anwenders, sein Programm an genau dieser oder jener Stelle zu unterbrechen. Benutzt man jedoch einen solches Breakpoint-Verfahren zur Herbeiführung eines Einzelschrittbetriebes wie weiter folgt, so ist es leider keineswegs trivial zu bestimmen, wo der Unterbrechungspunkt liegen muß.

Geht man davon aus, daß die Startadresse des Anwenderprogramms ersteinmal bekannt ist, so könnte man den unmittelbar folgenden Befehl mit einer Breakpoint-Instruction "patchen". Problem Nummer Eins: die Länge des aktuellen Befehls (in Byte) muß bekannt sein. Demzufolge ist eine Tabelle oder Berechnungsroutine anzulegen, die für alle möglichen Befehle die Länge enthält. Nun kann es sich aber immer noch um einen Sprungbefehl oder dergleichen handeln, bei dem die normale lineare Abarbeitung außer Kraft gesetzt wird. Neben Sprungbefehlen (incl. bedingter Sprünge) kommen auch Unterprogrammaufrufe oder die Rückkehr aus Unterprogrammen hinzu. Es bleibt nichts weiter übrig, als in einem solchen Fall das gesamte CPU-Verhalten softwaretechnisch vorauszuberechnen, um eine Vorhersage zu treffen, an welcher Adresse die Abarbeitung des Programms weitergehen würde. Diese Berechnungen können entweder auf dem Zielsystem ausgeführt werden (genügend Speicher und Rechenkapazität vorausgesetzt) oder aber auf einen PC (bzw. einer Workstation etc.) verlagert werden. Beide Varianten sind jedoch sehr aufwendig. Eine Realisierung dieser Methode ist z.B. in [6] am Beispiel eines Z80-Systems realisiert worden. Es ist eine sehr "saubere" Lösung, erfordert jedoch zu viel (Speicher-) Ressourcen, als daß eine Implementierung direkt auf dem HC11 in Frage käme.

3.7. Interrupterzeugung durch externe Hardware

Dieser Variante kommt einige Bedeutung zu, zumal der zu treibende Aufwand an extern zu realisierender Hardware überschaubar bleibt. Die Grundidee dieser Methode besteht darin, den Prozessor in den Interrupt-Zustand zu versetzen, und bei Rückkehr in das aufrufende (Anwender-) Programm nach genau einem abgearbeiteten Befehl wieder einen erneuten Interrupt auszulösen. Die Aufgabe der Interrupt-Generierung wird von einer externen Zählerschaltung realisiert. In der Übersicht könnte dies wie im folgenden Bild dargestellt realisiert werden:

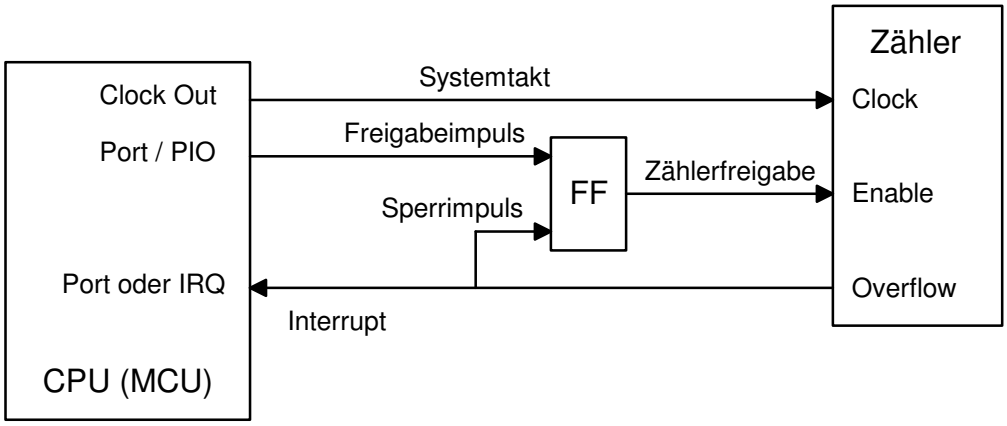

Bild 2: Singlestepbetrieb durch externe Interrupterzeugung

Zur Funktion: Der Prozessor befinde sich innerhalb einer Interrupt-Service-Routine. Es sei sichergestellt, daß sich die korrekten Registerwerte des zu testenden Anwenderprogramms, insbesondere der ProgramCounter und das Flag-Register, auf dem Stack befinden. Nun wird die Wiederaufnahme des Programms vorbereitet. Zuerst wird der externe Zähler freigegeben. Der im folgenden abgearbeitete Code muß stets die selbe Anzahl Taktzyklen erfordern. Jene werden vom Zähler mitgezählt. Befindet sich der Prozessor gerade bei Beginn der Abarbeitung des ersten Befehls des Anwenderprogramms, ist der Zählerendstand (bei Einsatz eines Rückwärtszählers: 0) erreicht und ein Interrupt wird ausgelöst. Nach Fertigstellung des Anwenderbefehls verzweigt der Prozessor (nach Retten der Register) wieder zur Interrupt-Service-Routine und der Vorgang kann wiederholt werden. Innerhalb der Interrupt-Service-Routine kann sinnvollerweise eine Meldung über den Prozessorzustand ausgegeben werden, dazu muß lediglich der Stack ausgelesen und in geeigneter Form visualisiert werden.

Die Nachteile dieses Lösungsansatzes liegen im Prinzip der Realisierung begründet: Ist etwa in dem zu testenden Programm ein Befehl enthalten, der die Interruptbehandlung sperrt (unterschiedliche Befehle sind denkbar, je nach Prozessor und Interruptsystem), so wird der folgende Versuch, einen Interrupt auszulösen, fehlschlagen.

3.8. Timer Interrupt

Mikrocontroller besitzen gegenüber einer CPU den Vorteil des (eingebauten) Vorrats an Peripheriefunktionen. Zu diesen Peripheriefunktionen zählt i.d.R. ein Timersystem. Daher liegt es nahe, das obige Beispiel der Einzelschrittabarbeitung mittels externem Zähler, bei Vorhandensein entsprechender interner Hardware, in die MCU zu verlagern. Ein solches Verfahren enthält zwar den Nachteil, daß der benutzte interne Timer nicht mehr frei verfügbar ist, jedoch kann Hardwareaufwand jeglicher Art eingespart werden. Gerade dieser Aspekt ist wichtig bei bereits bestehenden Hardwarelösungen, da solche Systeme oft nicht ohne beträchtlichen Aufwand modifiziert werden können. Vorausgesetzt es existieren ausreichend Timer (wie etwa im HC11), so ist die Reservierung eines dieser Timer um so leichter zu akzeptieren. Eine spezielle Realisierungsvariante dieses Verfahrens der Interrupt-Erzeugung mittels On-Chip-Timer ist Grundlage des im folgenden beschriebenen Kerns der vorliegenden Arbeit.

4. Einzelschrittbetrieb für die MCU 68HC11

4.1. Überblick: Special Bootstrap Mode, Timersystem, SCI, EEPROM

Um die Kommunikation zwischen PC und Mikrocontroller zu ermöglichen, sind insbesondere auf der Mikrocontrollerseite Vorkehrungen zu treffen. Dabei ist es ein wesentlicher Aspekt, daß der Mikrocontroller üblicherweise keine oder zumindest keine wesentlichen Hard- und Firmwarekomponenten besitzt, die auf eine High-Level Kommunikation im Debug-Modus ausgelegt sind. Solcherlei Eigenschaften findet man neuerdings erst bei Systemen aus dem 16/32 Bit Bereich, z.B. in Form der Background-Debug Schnittstelle der 68xxx-basierten Controllerfamilie Motorolas und weiterer Hersteller. Der überwiegende Teil der heute eingesetzten Mikrocontroller bietet lediglich die Möglichkeit, während der Testphase ein spezielles Monitorprogramm einzusetzen. Ein derartiges Monitorprogramm belegt einen (oftmals nicht unerheblichen Teil) des externen Adreßraumes und wird üblicherweise in Form eines EPROM-Bausteins in das Mikrocontrollersystem eingebettet. Für die zu erstellende und zu testende Anwedersoftware bedeutet das in der Regel kleinere bis erhebliche Einschränkungen. Insbesondere kann von einem allgemein gehaltenen Monitorprogramm nicht erwartet werden, daß alle erdenklichen Konfigurationen und Varianten für den Einsatz des verwendeten Mikrocontrollers erfaßt werden. Ein weiteres Problem ergibt sich, wenn die Mikrocontroller-Schaltung aufgrund zu nutzender interner Ressourcen überhaupt keinen externen Speicherbereich aufweist. In diesem Fall hilft nur der Einsatz eines die endgültige Version (mehr oder weniger vollständig) nachbildenden, jedoch mit externem Speicher ausgestatteten Entwicklungskits. Der Nachteil sind Zusatzkosten und Abweichungen von der angestrebten Ziellösung.

Der HC11 bietet zur Lösung der hier geschilderten Schwierigkeiten ein interessantes Konzept an. Über zwei Anschlüsse des Mikrocontrollers (MODA und MODB) läßt sich eine der folgenden vier Betriebsarten einstellen:

L	L	Special Bootstrap Mode
L	H	Single Chip Mode
H	L	Special Test Mode
H	H	Expanded Multiplexed Mode

Wird während eines Reset-Zyklus an den Pins MODA und MODB ein L-Signal erkannt, so läuft ein spezieller Mechanismus ab, der so nur im Special Bootstrap Mode durchgeführt wird:

1. Einblendung des Bootstrap ROM im Bereich $BF40 bis $BFFF mit dem Bootladerprogram.

2. Änderung des Reset-Vektors zu $BF40, dadurch Aufruf des Bootladerprogramms. Sämtliche Interrupt-Vektoren werden in den internen RAM-Bereich umgeleitet:

$00C4	Serial Communication Interface
$00C7	Serial Peripheral Interface
$00CA	Pulse Accumulator Input Edge
$00CD	Pulse Accumulator Overflow
$00D0	Timer Overflow
$00D3	Timer Output Compare 5
$00D6	Timer Output Compare 4
$00D9	Timer Output Compare 3
$00DC	Timer Output Compare 2
$00DF	Timer Output Compare 1
$00E2	Timer Input Capture 3
$00E5	Timer Input Capture 2
$00E8	Timer Input Capture 1
$00EB	Real Time Interrupt
$00EE	IRQ
$00F1	XIRQ
$00F4	Software Interrupt Instruction
$00F7	Illegal Opcode
$00FA	COP Fail (Watchdog Timer)
$00FD	Clock Monitor

3. Der Bootlader initialisiert das SCI (asynchrone serielle Schnittstelle) und sendet auf dem TXD-Pin ein Break-Zeichen (permanentes Low). Bei einer Taktfrequenz von 8 MHz (intern 2 MHz) arbeitet das SCI mit 7812 Baud.

4. Der Bootlader wartet nun, daß der Benutzer über das RXD-Pin ein Zeichen schickt.

5. Handelt es sich um das Zeichen $FF, so verzweigt der Bootlader in eine Routine, die im folgenden genau 256 Zeichen vom SCI erwartet. Diese Zeichen werden im internen RAM ab Adresse $0000 abgelegt und jedes dieser 256 Zeichen wird vom SCI als Echo zur Kontrolle zurückgesendet. Wurde das letzte Zeichen transferiert, springt der Bootlader direkt zur Adresse $0000 und überträgt damit dem eben geladenen Programm die weitere Kontrolle.

6. So der Benutzer eine Routine zur Programmierung des internen EEPROMs des HC11 implementiert, die sowohl eine Relozierung auf den Adreßbereich $0000 aufweist, als auch bezüglich der Länge die vorhandenen Kapazitäten nicht übersteigt, so liegt es nahe, mittels eines solchen Programms weitere Zeichen zu übertragen, und Schritt für Schritt in den internen EEPROM zu programmieren.

Die eben vorgenommene Darstellung berücksichtigt nicht alle Varianten und Möglichkeiten der Special-Bootstap Betriebsart des HC11. Sie zeigt jedoch, daß nur unter Nutzung der Signale TXD, RXD und Reset bereits eine vollständige Kontrolle des Speicherressourcen,

und damit - mittelbar - des gesamten Mikrocontrollers möglich ist. Als zusätzliche Hardware sind neben dem Mikrocontroller 68HC11 lediglich (bei Benutzung der üblichen RS232 Signalpegel) noch Pegelwandler V.24/TTL für die drei benutzten Signale notwendig. Da das SCI des HC11 ohne Handshake-Leitungen auskommt, liegt es nahe, eine der freigewordenen Signale für die Fernsteuerung des Reset zu nutzen. Das folgende Bild zeigt die komplette Anschaltung des HC11 an einen PC gemäß der realisierten Musterlösung:

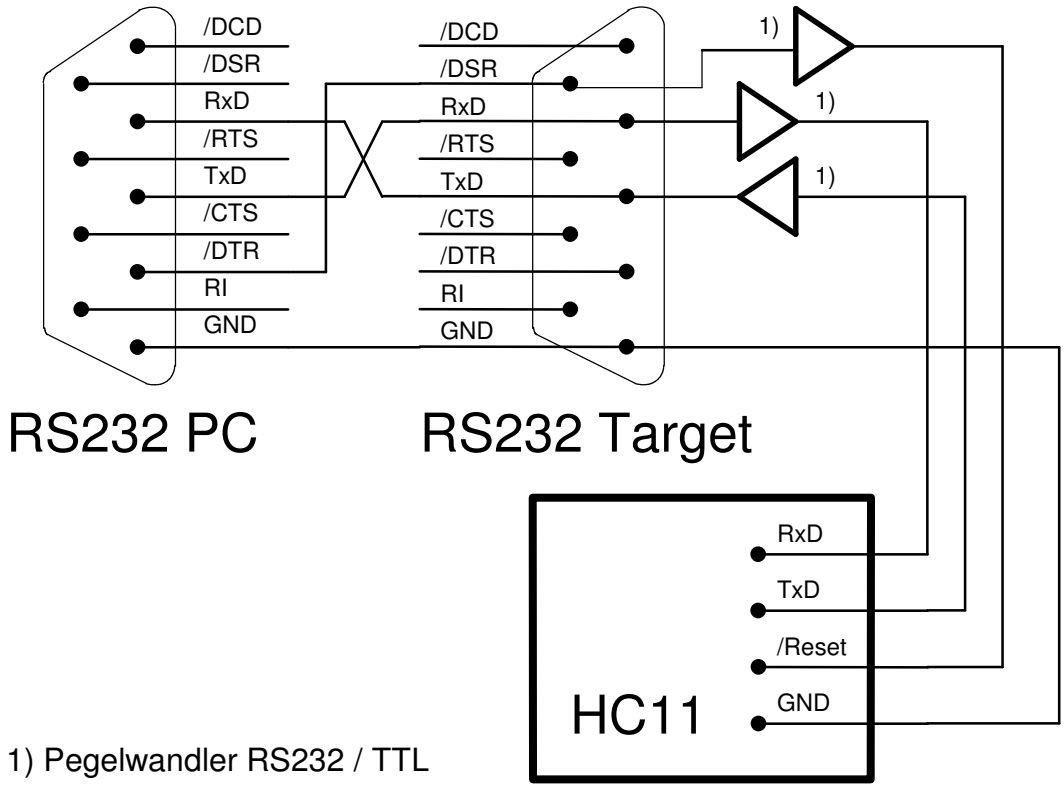

Bild 3: Verbindung zwischen PC und Zielsystem

4.2. Die Kommunikation zwischen PC und Zielsystem

Das obenstehende Bild zeigt alle notwendigen Verbindungen zwischen einem PC und dem HC11-Zielsystem auf.

Das Verbindungskabel zwischen den RS232-Schnittstellen des PC und des Target-Mikrocontrollers ist ein handelsübliches Nullmodemkabel. Durch die typische Kreuzung sowohl der Signal- als auch der Handshakeleitungen (im obigen Bild ist nur der relevante Teil dargestellt) ist das Reset-Signal des Target durch die Leitung /DTR (Data Terminal Ready, L-aktiv) der PC-RS232-Schnittstelle fernbedienbar.

Das Laden eines Programms in den RAM läuft, zusammengefaßt, gemäß der folgenden Darstellung ab:

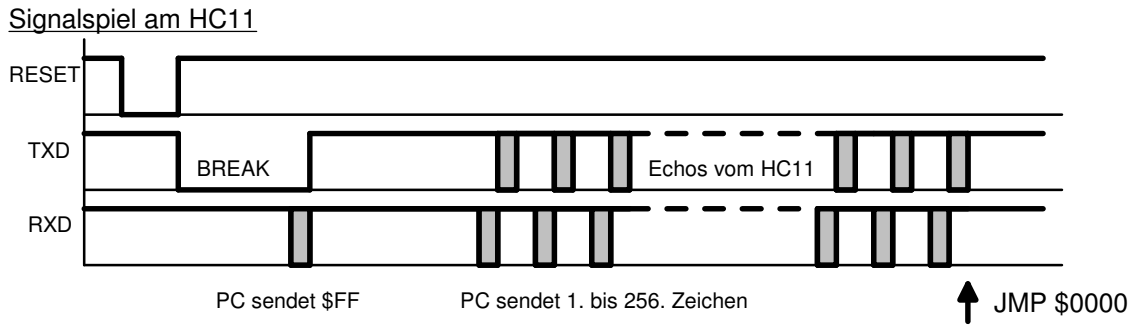

Bild 4: Kommunikationsprotokoll

4.3. EEPROMer Software für den HC11

Im Anhang B ist der Sourcecode für ein Programm zur Programmierung des HC11-internen EEPROMs angegeben. Dieses Programm ist für den im vorangegangenen Abschnitt geschilderten Bootlade-Mechanismus ausgelegt. Es wird, von der noch zu beschreibenden PC-Software unterstützt, nach einem Reset vom HC11 im Special Bootstrap Mode geladen und zur Ausführung gebracht.

Der 68HC11 verfügt über einen internen EEPROM-Bereich von 512 Byte im Falle des Typs 68HC11A1 bzw. 2048 Byte beim 68HC811E2. Die Lage des EEPROMs im Speicheradressraum der beiden Typen ist unterschiedlich. Bei dem ´A1-Typ beginnt der EEPROM auf Adresse $B600 und erstreckt sich bis $B7FF, während beim ´E2-Typ die erste EEPROM Adresse $F800 ist und die Endadresse bei $FFFF liegt.

Durch eine integrierte Ladungspumpe benötigt der HC11 keine zusätzliche, von 5 Volt abweichende Programmierspannung. Die Programmierspannung wird über das Bit 0 des Steuerregisters PPROG ein- bzw. ausgeschaltet. Bit 4 desselben Registers ist zuständig für die Auswahl des Programmiermodus. Steht dieses Bit auf 1, so bezieht sich der Schreib- bzw. Löschzugriff auf nur ein Byte. Ist das Bit 0, so ist der gesamte EEPROM-Bereich betroffen. Weiterhin von Bedeutung sind die Bits 1 und 2. Bit 1 steuert den Zugriff auf den EEPROM entweder als normale Leseoperation (Bit1=0) oder als Programmier- bzw. Löschzyklus (Bit1=1). Bit 2 schließlich spezifiziert, ob es sich um einen Programmierzyklus (Bit2=0) oder Löschzyklus (Bit2=1) handelt. Die anderen Bits haben nur untergeordnete Bedeutung (Testmode) und sind üblicherweise mit 0 belegt. Zur Übersicht die Belegung des PPROG-Registers:

Bit	Name	Funktion	Belegung 0	Belegung 1
0	EEPGM	Programmierspannung schalten	Aus	Ein
1	EELAT	EEPROM Zugriff	Lesen	Programmieren bzw. Löschen
2	ERASE	Lösch-Modus	Programmieren (bzw. Lesen)	Löschen
3	ROW	Zeile löschen (nur von Bedeutung, wenn BYTE=0)	gesamten EEPROM löschen	eine Zeile löschen
4	BYTE	Byte löschen	eine Zeile bzw. gesamten EEPROM löschen	ein Byte löschen
5	------	Nicht belegt	immer 0	nicht erlaubt
6	EVEN	Gerade Zeilen programmieren (Testmode)	Aus	Ein
7	ODD	Ungerade Zeilen progr. (Testmode)	Aus	Ein

Zusammengefaßt ergeben sich die folgenden relevanten Bitkombinationen für das PPROG-Steuerregister:

%00000000	$00	Normaler Lesezugriff
%00010110	$16	Byte löschen (vorbereiten) Das EEPROM-Byte, auf welches unmittelbar danach ein Schreibzugriff erfolgt (Dummy-Write), wird nach Zuschalten der Programmierspannung gelöscht.
%00010111	$17	Byte löschen (ausführen) sollte für min. 10 ms aktiv bleiben
%00000010	$02	Byte programmieren (vorbereiten) Ein nachfolgender Schreibzugriff auf ein EEPROM-Byte bewirkt (bei Zuschalten der Programmierspannung) die Programmierung des geschriebenen Datenbytes.
%00000011	$03	Byte programmieren (ausführen) sollte für min. 10 ms aktiv bleiben
%00000110	$06	gesamten EEPROM löschen (vorbereiten)
%00000111	$07	gesamten EEPROM löschen (ausführen) sollte für min. 10 ms aktiv bleiben

Die vorgestellte EEPROM-Programmiersoftware ist wie folgt implementiert (vergleiche dazu Listing im Anhang B!).

Das Programm beginnt gemäß den Anforderungen an eine Bootlade-Routine bei Adresse $0000. Das Setzen des Stackpointers auf $00FF und die Initialisierung des SCI wurde bereits

von der Bootstrap-Firmware des HC11 vorgenommen. Die Baudrate liegt bei 7812, ein interner Takt von 2 MHz vorausgesetzt.

Der zweimalige Aufruf der Unterfunktion "sciget" empfängt die Startadresse des zu programmierenden EEPROM-Bereiches. Die Reihenfolge: erst das niederwertige Byte, dann das höherwertige Byte. In Akku B wird der für das PPROG benötigte Wert geladen, das X-Register ist mit der Zieladresse initialisiert und Akku A enthält nach nochmaligem Aufruf der seriellen Empfangsfunktion "sciget" das zu programmierende Byte. Die Subroutine zur Programmierung eines Byte "eewrite" kehrt mit dem Wert zurück, der nach dem Programmierzyklus zurückgelesen wurde. Im Falle einer fehlerfreien Abarbeitung ist der Wert identisch mit dem zu programmierenden Byte. Zur Kontrolle wird dieser Wert durch das SCI zurückgesendet. Nach Erhöhung des Adreßzählers im X-Register wiederholt sich der Vorgang endlos - bis zum nächsten Reset.

Die "eewrite"-Subroutine vergleicht zuerst, ob das zu schreibende Byte bereits (zufällig) so im Speicher steht. Ist dies der Fall, so wird die Routine sofort verlassen. Speziell bei aufeinanderfolgenden EEPROM-Downloads mit nur geringfügig geändertem Programmcode ist es damit möglich, den gesamten Vorgang wesentlich abzukürzen. Liegt das Byte noch nicht in der erforderlichen Form vor, wird der Byte-Erase-Mode eingeschaltet und das Byte zuerst gelöscht. Eine Verzögerung von 10 ms ist für die korrekte Ausführung dieser Funktion notwendig. Das gelöschte Byte wird anschließend - nach Umschalten in den Programmier-Mode - mit dem gewünschten Wert beschrieben und wiederum innerhalb 10 ms sicher programmiert. Abschließend wird durch Löschen des PPROG-Registers zurückgeschaltet in den normalen Read-Mode und das gelesene Byte der aufrufenden Funktion zur Kontrolle zurückgeliefert.

Diese Schleife zur Programmierung von EEPROM-Zellen wird mit aufsteigenden Adressen fortgeführt, solange über das SCI Daten ankommen. Ein Abbruch der Schleife erfolgt durch das Auslösen eines Reset.

4.4. Trace Programm für den HC11

Ausgehend von der Implementierung des oben beschriebenen Algorithmus zur Programmierung des internen EEPROMs des Mikrocontrollers 68HC11 unter Nutzung des Special Bootstrap Mode entstand die Überlegung, die vorhandenen Möglichkeiten auch zur Realisierung eines Debugging-Algorithmus zu nutzen. Angestrebt wurde insbesondere die Einzelschrittabarbeitung eines Benutzerprogramms zu Testzwecken. Die auffälligste Einschränkung stellte zunächst der sehr knapp bemessene Speicherplatz für eine solche Software dar. Der gesamte EEPROM-Bereich sollte nach Möglichkeit dem Anwenderprogramm vorbehalten bleiben, in der praktischen Anwendung wird der Programmcode bis auf wenige Ausnahmen immer im EEPROM residieren. Kaum eine Applikationssoftware kommt jedoch ohne einige RAM-Zellen für Variable, Stack etc. aus. Daher sollte auch noch ausreichend Platz im RAM-Bereich verbleiben. Diese Prämissen stellten sehr harte Rahmenbedingungen dar, die eine erfolgreiche Realisierung des Singlestep-Moduls zuerst wenig aussichtsreich erschienen ließen.

Der erste untersuchte Weg zur Realisierung des Einzelschrittbetriebes bestand darin, jeweils das erste auf den aktuellen Befehl folgende Byte mit einer SWI-Instruction oder einem illegalen Opcode zu patchen. Dadurch wäre es möglich gewesen, in eine Interrupt-Routine zu

verzweigen, die Informationen zum aktuellen Zustand der MCU übermittelt. Dieses bereits im Abschnitt "Testmöglichkeiten für Mikrocontroller" beschriebene Verfahren erfordert jedoch die praktisch komplette Simulation der Prozessoraktivitäten auf der PC-Seite. In einem linearen Codeverlauf ist die Startadresse der folgenden Instruktion leicht über eine Tabelle zu ermitteln, die die Längen der einzelnen Befehle beinhaltet. Zum aktuellen Stand des Programcounters addiert man die Länge (in Byte) des abzuarbeitenden Befehls und erhält im Ergebnis die Location des nächsten Opcodes. Tritt jedoch eine Programmverzweigung auf, so ist man gezwungen festzustellen, wohin verzweigt wird. Besonders problematisch stellt sich die Verfolgung solcher Befehle wie bedingter Sprünge oder RTS (Return from Subroutine) dar. Der Aufwand für eine solche umfangreiche Simulation der Target-MCU auf dem PC erschien unangemessen, da vorhandene reine Softwaresimulatoren diese Aufgabe prinzipiell ebenso erfüllen; zumal ohne das Vorhandensein einer realen Hardware.

Das komplexe Timersystem des HC11 eröffnet jedoch eine weitere Möglichkeit für die Realisierung des Einzelschrittbetriebes. Trotz der vielschichtigen Komplexität des Timersystems ist die Benutzung nicht auch zwingend Codeintensiv. Dieser Umstand kommt der angestrebten speicherplatzsparenden Lösung sehr zugute. Mittels des im folgenden vorgestellten kompakten Bootlader-Programms gelang nicht nur die Realisierung hinsichtlich der Aufgabenstellung, sondern es verblieben dem Anwender auch maximale Freiheiten bezüglich der weiterhin zur Verfügung stehenden Ressourcen des Mikrocontrollers, insbesondere die Speicherressourcen betreffend.

Zur Erläuterung vorab einige Bemerkungen zum Timersystem. Basis des Timersystems ist ein kontinuierlich laufender 16-Bit Zähler. Nach Reset wird dieser Systemzähler durch den internen E-Clock (üblicherweise 2 MHz) gespeist. Es existiert ein Vorteiler, der (in den ersten 64 Taktzyklen nach Reset) auf eines von drei Teilerverhältnissen eingestellt werden kann. In der hier beschriebenen Lösung muß der Vorteiler unverändert bleiben (Teilerverhältnis "1:1"); dies ist auch im stark überwiegenden Teil aller Anwendungen sinnvoll. Damit erhält der Systemzähler aller 500 ns einen Zählimpuls. Läuft der Systemzähler über, kann bei Bedarf ein Interrupt ausgelöst werden. Ein 16-Bit Lesezugriff auf den Systemzähler wird intern so synchronisiert, daß stets der gültige Wert ermittelt wird; ein Fehler durch einen auftretenden Übertrag von L-Byte auf H-Byte während des Lesens ist ausgeschlossen.

Der HC11 besitzt fünf sogenannte Output Compare Register. In Ihnen kann jeweils ein 16-Bit Wert abgelegt werden. Stimmt der Stand des Systemzählers mit dem Inhalt eines dieser Output Compare Register überein, wird ein Flag gesetzt und bei Bedarf ein Interrupt ausgelöst. Jeder Output Compare Kanal besitzt ein eigenes Flag und einen korrespondierenden Interrupt Vektor. Weiterhin kann jeder Output Compare Kanal auf einem zugehörigen Ausgangspin des HC11 einen Pegel (-wechsel) erwirken.

Zur Interruptfreigabe ist jedem OC Kanal ein Bit im Register TMSK1 zugeordnet:

TMSK1 Timer Interrupt Mask Register 1 ($1022)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OC1I	**OC2I**	**OC3I**	**OC4I**	**OC5I**	IC1I	IC2I	IC3I

Gleichartig aufgebaut ist das Flag Register, es enthält für jeden OC Kanal ein Bit zur Signalisierung eines anstehenden Interrupts:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OC1F	**OC2F**	**OC3F**	**OC4F**	**OC5F**	IC1F	IC2F	IC3F

Die Lage der Interruptvektoren wurde bereits im Abschnitt "Special Bootstrap Mode" beschrieben. Vorgesehen sind jeweils 3 Byte (einen JMP-Befehl aufnehmend) ab Adresse $00D3 (OC5) bis $00E1 (letztes Byte für OC1).

Der Schlüssel zur Realisierung des Einzelschrittbetriebes liegt nunmehr in der Funktionsweise der Interruptgenerierung des HC11. Interrupts bleiben (abgesehen von XIRQ) gesperrt, solange das I-Bit im CCR (Flagregister der MCU) gesetzt ist. Wird z.B. durch den Befehl CLI das I-Bit rückgesetzt, so prüft die MCU vor Abarbeitung jedes Befehles, ob ein Interrupt anhängig ist. Trifft dies zu, wird automatisch (nach dem Retten aller Register auf den Stack) das I-Bit gesetzt und somit Interrupt-Verschachtelungen standardmäßig vorgebeugt. Nach der Feststellung der Interruptquelle (bei mehreren gleichzeitig aktiven: die höchstpriorisierte) wird der zugehörige Interruptvektor in den Programcounter geladen und im Special Bootstrap Mode zu dem im RAM-Bereich vom Benutzer abgelegten JMP-Befehl zur gewünschten Interrupt Service Routine verzweigt. Nach Beendigung dieser Interrupt Service Routine (abschließender Befehl: RTI) werden die Register wieder vom Stack geholt. Da dies auch für das CCR-Register zutrifft, und dieses das unveränderte (den Wert 0 beinhaltende) I-Flag restauriert, werden Interrupts automatisch wieder zugelassen.

Hält sich der Prozessor permanent in einer Interrupt-Service Routine auf, die nur auf ein Benutzerkommando hin verlassen wird, und wird vor Verlassen der ISR ein Timer Output Compare Register derart gesetzt, daß innerhalb des ersten Befehls nach dem die ISR beendenden RTI-Befehl ein Interrupt ausgelöst wird, so ist jener Befehl die einzige Instruktion, die abgearbeitet wird vor Wiedereintritt in die Interrupt Service Routine. Dieses Verfahren ist in der realisierten Art und Weise neuartig, insbesondere im Zusammenspiel mit dem Special Bootstrap Mode des HC11.

Das folgende Flußdiagramm zeigt die Verhältnisse nochmals grafisch:

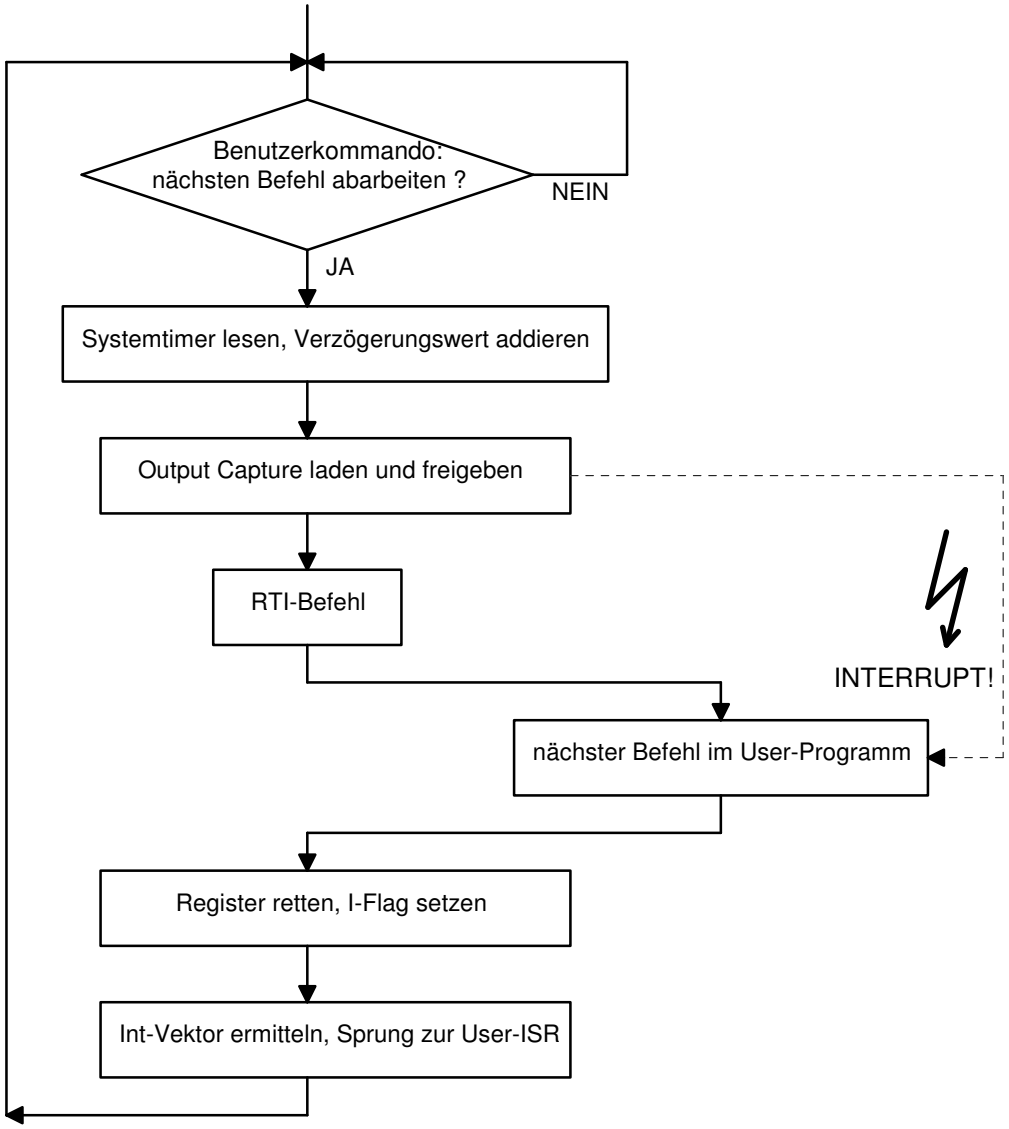

Bild 5: Einzelschrittbetrieb per Output Compare Interrupt

Das Verfahren wurde selbstverständlich noch um Kommunikationselemente bereichert, die einen Rückschluß auf den Zustand der MCU zulassen. Das im Anhang C abgedruckte Sourcelisting zeigt die Details.

Die Interrupt-Service Routine startet am Label "isr". Die Registerinhalte des Anwenderprogramms wurden durch den Interruptzyklus auf dem Stack gerettet. Die Interrupt Service Routine liest diese Registerinhalte aus und überträgt sie über das SCI zum auswertenden PC. Außerdem werden aus dem Speicher die unmittelbar folgenden neun Codebytes (wobei die Länge einer HC11-Instruction inklusive aller Operanden maximal 5 Byte beträgt) ausgelesen und gesendet. Schließlich folgen die Belegungen der HC11-Ports A bis E. Die von der CPU gelesenen Werte werden auch über das SCI gesendet. Ein Synchronisationsbyte ($0A) schließt das Datentelegramm ab:

<PC+0>
ACCB
<PC+1>
ACCA
<PC+2>
IXH
<PC+3>
IXL
<PC+4>
IYH
<PC+5>
IYL
<PC+6>
PCH
<PC+7>
PCL
<PC+8>
SPH
SPL
PORTA
PORTB
PORTC
PORTD
PORTE
$0A

Die Instruktionen am Ende der ISR bereiten die Abarbeitung des nächsten Befehls im Anwenderprogramm vor. Wichtig ist hierbei die genaue Kenntnis der resultierenden Verzögerungszeit, schließlich soll der Timerinterrupt genau im ersten Takt nach Beginn jenes ersten Befehls eintreten. Die letzten 6 Befehle der ISR sind folgend mit ihren jeweiligen Befehlslängen (in Taktzyklen, ein Taktzyklus = 500 ns bei einem internen Takt von 2 MHz) dargestellt. Zur Interruptgenerierung wurde der Timer Output Capture 5 verwendet:

Befehl	Länge	Länge gesamt
`ldd otcnt,x`	1 (anteilig)	1
`addd #33`	4	5
`std otoc5,x`	5	10
`ldaa #$08`	2	12
`staa otmsk1,x`	4	16
`staa otflg1,x`	4	20
`rti`	12	32

Bisher wurde vorausgesetzt, daß sich die MCU bereits innerhalb der ISR befindet. Selbstverständlich ist klar, daß ein Startup-Code vorhanden sein muß, um diesen Zustand zu erreichen, und den allerersten Befehl des Anwenderprogramms abzuarbeiten. Der Startup-Code ist bezüglich der OC5-Vorbereitungen fast identisch mit dem eben beschriebenen Programmstück der ISR. Zusätzlich wird jedoch die lokale Interruptmaske des Output Compare 5 Interrupts freigegeben und die globale Interruptfreigabe per CLI-Befehl bewirkt. Schließlich wird ein Sprung zur gewünschten Anfangsadresse ausgeführt. Während dieses Sprungbefehls wirkt der OC5-Interrupt erstmalig und bewirkt einen Programmstop vor dem ersten Befehl des Anwenderprogramms.

Adreßlage, Initialisierung und Übertragung der gewünschten Anfangsadresse ist wie im bereits oben geschilderten EEPROM-Programmierprogramm realisiert. Neben zwei Unterroutinen zum Senden bzw. Empfangen eines Zeichens über das SCI ("sciput" und "sciget") ist lediglich noch das Label "OC5VECT" zu bemerken. An der Adresse $00D3 ist mit diesem Label ein Sprung zur Interrupt Service Routine eingetragen.

4.5. Verfahrensbedingte Einschränkungen

Die Anwendung eines Programmier-Tools erfordert stets auch die genaue Kenntnis der Einschränkungen in der Nutzung vorhandener Befehle, Verfahren oder Ressourcen. Bei dem HC11 Singlestep-Debugger sind, wenn das Anwenderprogramm zum ersten mal aufgerufen wird, bereits diverse MCU-Funktionen initialisiert. Verursacher dafür ist einerseits die Firmware (Bootstrap-Lader) und andererseits das primär geladene Trace-Programm. Wichtige Randbedingungen für das auszutestende Anwenderprogramm sind folgende Punkte:

Der Stack ist initialisiert auf $00FF (das Ende des internen RAM´s). Der Interrupt-Pseudo-Vektor für den Output Compare Kanal 5 ist besetzt, damit hat der Stack eine maximale zulässige Ausdehnung bis herab zur Adresse $00D6 (incl.). Zu beachten ist die Stackbelastung durch die Interrupt Service Routine mit 14 Bytes. Somit verbleiben im Standardfall 28 Byte Stack für das Anwenderprogramm, was einen akzeptablen Wert darstellt. Es ist allerdings jederzeit zulässig, den Stackpointer auf einen anderen (freien) Bereich zu setzen.

Der Timer Prescaler (Bits PR0, PR1 im Register TMSK2) ist auf 0:0 gesetzt, ein Timer Tick entspricht damit 500 ns. Ein Anwenderprogramm kann die Prescaler-Bits beschreiben, jedoch wirken sich derlei Änderungen nicht aus, die die Interrupt Service Routine das Register TMSK2 jedesmal reinitialisiert.

Das SCI (Serial Communication Interface) ist auf 7812 Baud initialisiert und freigegeben. Die Kommunikationsparamter des SCI dürfen nicht verändert werden, dazu gehört insbesondere auch der Vorteilerwert. Zeichenempfang und -aussendung werden im Polling betrieben.

Über das HPRIO Register wird dem verwendeten OC5-Interrupt die höchste Priorität eingeräumt. Das Anwenderprogramm kann diese Priorität nicht ändern, da Schreibzugriffe auf die Priority Select Bits PSEL0...3 im HPRIO Register nur bei gesetztem I-Flag möglich sind, das I-Flag jedoch innerhalb des Anwenderprogramms stets Null sein muß. Dennoch sollten Schreibzugriffe auf deas HPRIO Register vermieden werden, da versehentlich über das RBOOT Bit der Bootstrap ROM ausgeblendet werden könnte. Dies führt zum Verlust aller (primärer) Interrupt Vektoren und damit zur Fehlfunktion des OC5 Interrupts.

Der Befehl SEI setzt das I-Flag und verhindert dadurch die Annahme des Trace Interrupts. Der Befehl SWI und jeder illegaler Opcode haben die gleiche Auswirkung.

Der Befehl WAI wirkt wie ein NOP Befehl, da unmittelbar nach Ausführung dieser Instruction ein Interrupt (der Trace Interrupt, OC5) anhängig ist, welcher nach Retten der Register (ordnungsgemäß) in die ISR des Trace-Mode verzweigt.

Die STOP Instruction führt, in Verbindung mit einem rückgesetzten S-Flag (STOP Disable Flag), zum Einfrieren der MCU-Aktivitäten und ist daher nicht sinnvoll im Einzelschrittbetrieb testbar. Ist das S-Flag gesetzt (default), so wirkt STOP wie ein NOP Befehl.

Das Beschreiben der Interrupt Pseudo Vektoren (im RAM) ist nicht zulässig, da die entsprechenden Speicherbereiche anderweitig belegt sind. Das nachstehende Bild zeigt die Aufteilung des RAM-Adreßraumes im Trace-Mode:

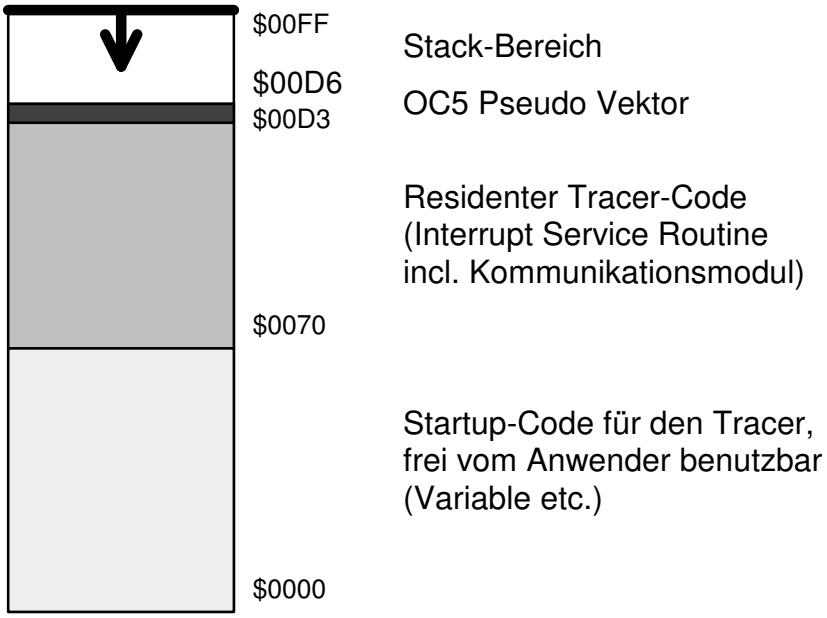

Bild 6: Speicherbelegung im Trace-Mode

Der Bereich von $0000 bis $006F ist nur in der Initialisierungsphase (vor dem ersten Programmschritt) von Bedeutung, er kann ansonsten beliebig benutzt werden.

Probleme kann auch der Befehl TAP bereiten. Obwohl dies nicht auf den ersten Blick ersichtlich ist, kann das System nach einem TAP-Befehl "hängen". Ursache dafür ist das versehentliche Setzen des I-Flag, was einem SEI-Befehl gleichkommt. Die globalen Sperrung der Interrupts wäre das Ergebnis, und damit eine Außerbetriebsetzung des Trace-Mode. Die RTI Instruction kann unter Umständen denselben Effekt hervorrufen.

5. Die integrierte Entwicklungsumgebung

Der folgende Abschnitt erläutert die Art und Weise der Implementation der integrierten Entwicklungsumgebung (HC11 Integrated Development Environment, kurz HC11IDE), welche letztlich das Benutzerinterface für Anwender des oben beschriebenen Einzelschritt-Verfahrens darstellt. Hardwaregrundlage ist für die folgenden Ausführungen und die beschriebene Software stets ein IBM-kompatibler PC der AT-Kategorie.

Die Software trüge nicht in der Bezeichnung das Attribut "integriert", wenn der Einzelschrittbetrieb die einzige zu realisierende Funktion wäre. Über die Vielzahl von Funktionen des Programms informiert Sie der Anhang A mit dem umfangreichen Benutzerhandbuch, an dieser Stelle wird die Kenntnis der wichtigsten Menüpunkte und Programmteile vorausgesetzt.

Als weitere wesentliche Voraussetzung zum Verständnis der hier vorgestellten PC-Software ist, neben der Vertrautheit mit dem Borland C++ Compiler, die Kenntnis der Struktur und "Philisophie" der Turbo Vision Library zu nennen. Diese Bibliothek zur Realisierung ereignisgesteuerter, objekt- sowie fensterorientierter Programme, welche hier in der C++ Version eingesetzt wurde, ist der Schlüssel zur Gestaltung eines homogenen, benutzerfreundlichen Anwenderprogramms mit Schwerpunkt auf der funktionellen Implementation. Anstatt 80 Prozent und mehr auf die Codierung von Menüs, Statuszeilen, Bildschirmausgaben etc. zu verwenden, kann man das Gerüst der Turbo Vision API nutzen und alle wesentlichen Kommunikationselemente "erben". Mehr zur Programmierung mit der Turbo Vision Library erfährt der Leser in [9] und [10].

5.1. Überblick über die HC11IDE-Software

Die HC11IDE Software besteht aus mehreren Programmkomponenten. Jede dieser Komponenten basiert auf einer oder aber mehreren Quelltextdateien. Das Hauptprogramm HC11IDE.EXE wird gebildet aus den Quelltexten:

- HC_MAIN.CPP
- HC_DLG.CPP
- MC_MENU.CPP
- HC_TERM.CPP
- HC_REAS.CPP
- HC_SR2B.CPP
- HC_HEAP.CPP
- FILEVIEW.CPP

sowie den beiden Header- und der Projectdatei:

- HC11IDE.H
- FILEVIEW.H
- HC11IDE.PRJ

Das Hauptprogramm wird, zusammen mit der Turbo Vision Library TV.LIB und den Standardbibliotheken, für das Speichermodell LARGE übersetzt. Alle weiteren Compilereinstellungen sind in der Projectdatei festgehalten.

Daneben existieren einige Hilfsprogramme. Das Programm ERR_FIND.EXE wird definiert durch den Quelltext ERR_FIND.C. Die beiden Tools A!.BAT und S!.BAT sind Stapeldateien zum Aufruf der externen Programme Assembler (ASMHC11 Freeware Assembler der Firma Motorola) und Simulator (TESTE68 Befehlsinterpreter der Firma Rohde Interface). An dieser Stelle sei ausdrücklich angemerkt, daß es sich bei letzgenannter Software keinesfalls um Freeware oder Public Domain Software handelt. Das Programm TESTE68 ist ein kommerzielles Programm, welches separat käuflich zu erwerben ist und nicht standardmäßig zum Ausstattungsumfang der vorliegenden HC11IDE Software gehört.

5.2. Die Aufgaben der Quelltextmodule

5.2.1. Headerdatei - HC11IDE.H

Diese Headerdatei enthält alle benutzerdefinierten Klassen bzw, Strukturen, die Konstantenvereinbarungen, Funktionsprototypen, Präprozessordefinitionen und alle weiteren benötigten (System-) Include-Dateien.

Am Beginn der Datei sind per #define-Anweisung alle verwendeten Klassen der Turbo Vision Library (kurz: TVL) aufgeführt. Die unmittelbar folgende Headerdatei TV.H (Bestandteil der TVL) greift auf diese Definition zurück und stellt ihrerseits alle benötigten Prototypen jener Klassen zur Verfügung. Es folgen die #include-Anweisungen für eine Reihe von Headerdateien des Borland C++ Compilers.

Im Anschluß daran erfolgen einige Definitionen, und zwar für Bitmasken sowie Fehlercodes des Kommunikationsmodul. Die folgenden Konstanten ergänzen die cmXXX-Konstanten der TVL um benutzerdefinierte Command-Konstanten.

Unter function prototypes sind Funktionsprototypen für all jene Funktionen aufgeführt, die nicht Memberfunctions einer Klasse sind. Der überwiegende Rest der Headerdatei ist belegt mit den class- und struct-Typdefinitionen der HC11IDE.

5.2.2. Hauptbestandteile - HC_MAIN.CPP

Diese Datei enthält die wesentlichsten Bestandteile des Programms. Neben der abgeleiteten Applikationsklasse HC11IDEapp sind dies die Klassen UErrViewer (UserclassErrorViewer), UTraceView (UserclassTraceinfoView) und UEditWindow (UserclassEditorWindow).

Im Quelltext dieses Moduls schließt sich an die obligatorischen Abschnitte "defines", "includes" und "globals" die Klass UErrViewer an. Diese Klasse stammt im Wesentlichen von TWindow ab, enthält aber ein (View-) Objekt der Klasse TFileViewer, welches die Darstellung einer Textdatei im Fenster ermöglicht. Im Gegensatz zu TEditor sind jedoch keine Editorfunktionen enthalten, sondern nur die Anzeige der Datei ist möglich.

Die Elementfunktion handleEvent() der Klasse UErrViewer überschreibt die entsprechende Basisklassen-Funktion, um die Umsetzung der Eingabe von <ESC> in ein cmClose Kommando vorzunehmen. Dies führt zum Schließen des UErrViewer Fensters. Normalerweise ist dieses nur über Eingabe von ALT-F3 möglich, die Berücksichtigung von <ESC> dient der Vereinfachung der Bedienung, zumal das Fehlerfenster nur jeweils kurzzeitig betrachtet wird.

Die close() Elementfunktion ist ebenfalls überschrieben, um die Position des Fehlerfensters festzuhalten. Sie wird in der globalen Variablen rErrWin (Typ TRect) festgehalten, und steht bei einer erneuten Öffnung eines Fehlerfensters wieder zur Verfügung. Dadurch befindet sich das Fehlerfenster, hat es der Benutzer erst einmal positioniert, immer wieder an der gleichen Stelle auf dem Desktop.

Es folgen Elementfunktionen der Klasse UTraceView, welche den Inhalt jenes Fensters bilden, mit dem die Informationen während des Einzelschrittbetriebes angezeigt werden.

Der Konstruktor übergibt lediglich die Information zur Größe an den Konstruktor der Basisklasse TView() und generiert einen Aufruf der Elementfunktion update (Beschreibung siehe unten).

Die beiden Funktionen ito2x() und ito8b() sind Hilfsfunktionen zur Umsetzung eines Integer-Wertes in einen String. ito2x() generiert einen String mit 2 Hex-Digits, während ito8b() einen String mit 8 Binärziffern erzeugt. Die Funktionen wurden aufgrund der höheren Ausführungsgeschwindigkeit gegenüber sprintf() eingesetzt. Sie werden zum Darstellen der Trace-Informationen innerhalb der Klasse UTraceInfo verwendet.

UTraceView::draw() überschreibt die Funktion TView::draw() und sorgt somit für die gewünschte Bildschirmausgabe aller Trace-Informationen.

UTraceView::handleEvent() reagiert, zusätzlich zu den Funktionen von TView::handleEvent(), auf das Kommando cmTrace. Es wird, wie schon im Konstruktor, die Elementfunktion update() aufgerufen, welche per UTerminal::step4step() vom Kommunikationsmodul einen Programmschritt ausführen läßt. Die erhaltenen Informationen werden in der Struktur traceInfo abgelegt und per drawView() (impliziert den Aufruf von draw()) angezeigt.

Die Klasse UEditWindow wurde von der Turbo Vision Klasse TEditWindow abgeleitet. Die Funktionalität ist bei beiden Klassen praktisch identisch. Um jedoch die Möglichkeit einer Identifizierung zu erhalten, regiert die handleEvent() Funktion auf den Request cmWhoIsThere. Dadurch ist es der Routine, die diesen Request auf die Reise schickt, möglich, zu erfahren, ob ein UEditWindow-Objekt gerade den Fokus hat (also "oben" liegt, sichtbar ist). Verwendung findet dieses Feature bei den Assemlerfunktionen, um zu entscheiden, ob ein, und wenn ja, welcher Quelltext bzw. welches Editorfenster bearbeitet werden soll. Da clearEvent() einen Zeiger auf jenes UEditWindow-Objekt hinterlegt, welches clearEvent() aufgerufen hat, läßt sich dadurch ermitteln, wie die mit diesem Fenster verbundene Datei heißt. Verwendung findet der Mechanismus in HC11IDEapp::make(). Die handleEvent() Funktion reagiert weiterhin auf die Message cmUpdate, um die (nachstehende beschriebene) Funktion updateCommands() aufzurufen. Damit, und mit der überschriebenen Funktion shutDown(), werden

die Kommandos ein- bzw. ausgegeblendet, auf die das Editorfenster Einfluß hat (Beispiel: cmAssemble ist nur anwendbar, wenn ein Editorfenster geöffnet ist, sonst ist das Kommando disabled).

Hier schließen sich die Elementfunktionen der Applikationsklasse HC11IDEapp an. Der Konstruktor (an zweiter Position) nutzt die Konstruktoren der Basisklassen TProgInit und TApplication zur Initialisierung. Weiterhin werden die Menüpunkte, die zu Beginn der Programmausführung noch nicht zur Verfügung stehen, mittels disableCommands ausgeblendet (sie erscheinen damit vorerst grau in Menü und Statuszeile). Der Zeiger editorDialog der Klasse TEditor wird auf die implementierte, benutzerspezifische doEditDialog() Funktion gesetzt. Anschließend wird eine Hauptspeicheranzeige-View (THeapView) rechts unten in den Applikationsbildschirm eingefügt. Ähnliches geschieht mit dem Fenster des Clipboard (Zwischenablage für den Editor), jedoch ist dieses anfänglich verborgen, wird also erst nach einer entsprechenden Benutzereingabe sichtbar. Nach dem Initialisieren einiger System- und Benutzervariablen und der Erzeugung eines Kommunikationsobjekts der Klasse UTerminal (wird auf 7812 Baud und COM1 initialisiert) endet der Konstruktor.

Die Funktion HC11IDEapp::idle() wurde überschrieben, um die Hauptspeicher-Anzeige und das Kommandoset fortlaufend (wenn sonst keine Aktivitäten zu verzeichnen sind) zu aktualisieren.

HC11IDEapp::fileOpen() und HC11IDEapp::fileNew() werden über die gleichlautenden Menüpunkte aufgerufen. Sie verwenden beide die Elementfunktion openEditor() (Definition oberhalb des Konstruktors), welche ein neues Editorfenster anlegt.

Die nachstehenden Elementfunktionen der Klasse HC11IDEapp sind selbsterklärend und dienen jeweils zur Ausführung eines entsprechenden Menü-Kommandos. Die Zuordnung der Kommandos zu den einzelnen Elementfunktionen ist aus handleEvent() ersichtlich.

Die Funktion HC11IDEapp::make() ist etwas umfangreicher. Sie beinhaltet sowohl die notwendigen Bestandteile zur Ausführung der Kommandos cmAssemble, cmRebuild, cmExecute sowie cmSimulate. Zu Beginn wird das fokussierte Editorfenster ermittelt. Ist kein Editorfenster geöffnet, oder hat ein anderes Bildschirmelement den Fokus, so wird die Funktion mit der Fehlermeldung "Nothing to do" abgebrochen. Konnte jedoch ein Editorfenster gefunden werden, so wird der Name der darin bearbeiteten Quelltextdatei ermittelt und der Inhalt wird gespeichert, wenn ungespeicherte Änderungen im Quelltext vorhanden sind. Schließlich werden noch die Namen der Objekt-, Fehler und Binärdatei aus dem Namen der Quelldatei gebildet.

Ist das Fehlerfenster noch geöffnet, so wird es jetzt geschlossen. Anschließend wird der Assembler (extern) aufgerufen. Dem Assembler wird der Dateiname und der Wert der Assemblerkonstanten PROG mitgeteilt. Existiert nach dem Assemblieren eine Datei mit dem aktuellen Dateinamen und der Endung .ERR, so sind Fehler aufgetreten. Der erste Fehler ist in jener Fehlerdatei textuell aufgeführt und wird in einem Fehlerfenster angezeigt, nachdem die Meldung "Assembler error(s)" ausgegeben wurde. War die Assemblierung fehlerferei, so wird das entstandene Motorola-S-Record File (.OBJ) in eine Binärdatei (.BIN) umgewandelt. Dazu dient die Funktion sr2bin(). Die dabei möglicherweise auftretenden Fehler sind weiter unten beschrieben.

Außer bei dem Kommando cmRebuild kann der vorangegangene Abschnitt genau dann übersprungen werden, wenn bereits ein aktueller Assembleroutput inklusive Binärdatei vorliegt. Dies wird über die Dateierstellungszeit entschieden. Ist die Quelle neuer als eine vorhandene Binärdatei, muß neu übersetzt werden, ansonsten nicht - es erscheint die Meldung "Bin file is up to date", wenn es sich um das Kommando cmAssemble handelte.

Geschah bis zu diesem Zeitpunkt kein Abbruch, dann wird bei Ausführung der Kommandos cmAssemble bzw. cmRebuild die Erfolgsmeldung "Assembler: Success." ausgegeben und die make() Funktion damit beendet.

Wird hingegen das Kommando cmSimulate ausgeführt, so wird nun der (externe) HC-Simulator "TESTE68" aufgerufen. Dies geschieht, wie schon beim Assembler, über eine Stapelverarbeitungsdatei. Mit der Meldung "Simulator closed" wird nach Beendigung des Simulatorprogramms wieder zurückverzweigt.

Es bleibt noch die Möglichkeit, daß es sich um das Kommando cmExecute handelt, welches sich in Bearbeitung befindet. Hier muß unterschieden werden, wohin die Binärdatei geladen werden soll. Einerseits kann es sich um den EEPROM handeln, andererseits ist die Möglichkeit vorhanden, das Programm in den RAM zu laden. Die vom Benutzer eingestellte Varainte steht in der Variablen TargetOptions zur Auswertung zur Verfügung. In beiden möglichen Fällen wird zuerst ein Statusfenster erzeugt, das dem Benutzer bekanntgibt, was zur Zeit gerade geschieht. Über Elementfunktionen des Kommunikationsmoduls wird danach entweder der EEPROM programmiert, oder der RAM geladen. Bei Erfolg wird das geladene Programm zur Ausführung gebracht und die make() Funktion endet mit der Mitteilung "Target: Executing".

Die folgende Elementfunktion trace() geht davon aus, daß sich die abzuarbeitende Codesequenz bereits im Zielspeicher befindet. Nachdem das Trace-Programm in den HC11-RAM geladen wurde, wird ein Trace-Dialog erzeugt. Dies schließt die Erzeugung der Trace-View und die Bereitstellung der Trace-Information über das Kommunikationsmodul mit ein. Solange der Benutzer kein cmCancel Kommando erzeugt (<ESC>-Taste gedrückt), wird immer wieder ein Programmschritt abgearbeitet.

Für die Realisierung der Menüfunktion Hardware/Options steht die Elementfunktion targetOpts() zur Verfügung. Wünscht der Anwender eine Änderung der Hardwareoptionen, so werden als Resultat in dieser Funktion einige globale Variable gesetzt und eine Message abgesetzt, die für das Editorfenster bestimmt ist. Das Editorfenster kennzeichnet sich selber als modifiziert, um schließlich bei einem Aufruf vom make() eine Neuübersetzung zu erzwingen.

Zur about() Funktion ist wenig zu sagen, es wird lediglich ein Dialog angezeigt, der über die Urheberschaft und Versionsnummer der Software informiert.

In der Elementfunktion handleEvent() ist die Zuordnung der Kommandos (cmXXXX) zu den Behandlungsroutinen realisiert. Mehr als 5...6 Codezeilen sind nur für cmTraceFrom implementiert, hier wird zuerst über einen Dialog die gewünschte Startadresse für den Trace-Mode abgefragt, bevor die Funktion trace() aufgerufen wird.

In der main() Funktion wird, nachdem per analyze_commandline() die evtl. übergebenen Kommandozeilenparameter gescannt werden, lediglich eine Instanz der eben ausführlich besprochenen Klasse HC11IDEapp erzeugt und die Elementfunktion run() gerufen. main() kehrt stets mit Fehlercode Null zurück.

5.2.3. Dialogelemente - HC_DLG.CPP

Dialogelemente sind ein sehr wichtiger Bestandteil einer Applikation, denn hier wird der Benutzer mit mehr oder weniger komfortablen Möglichkeiten der Einflußnahme auf das Programm konfrontiert.

Der Vorteil der Turbo Vision Library kommt hier voll zum tragen. Die Erzeugung von Dialogen (Fenster die zwingend den Fokus besitzen, also nicht "abgewählt" werden können zu Gunsten anderer View-Objekte) verläuft immer nach dem gleichen Muster. Zuerst wird ein Dialog-Objekt erzeugt, welches anfangs leer ist. Anschließend wird der Dialog mit den zur Kommunikation erforderlichen View-Objekten gefüllt. Dies sind im wesentlichen Buttons, Eingabefelder, Auswahlfelder oder statische Textelemente.

Die erste im Quelltext aufgeführte Funktion execDialog() ist eine globale Funktion. Sie ist von jedem Programmteil aus ausführbar. Neben der Aufgabe, einen Dialog dynamisch zu erzeugen, und ihn nach Verlassen wieder aus dem Speicher zu entfernen, organisiert execDialog() auch die Übergabe und Rückgabe von Daten des Dialogs. Die Art und Weise der Datenübergabe ist universell gelöst, es wird einfach ein Pointer auf eine komplette Datensammlung übergeben. Details des Verfahren sind in [9] beschrieben.

Der Rest des Quelltextes, und damit der weitaus überwiegende Teil, ist mit den einzelnen Funktionen zur Erzeugung der Dialoge des Programms belegt. Die Definition der einzelnen Dialogelemente ist im wesentlichen selbsterklärend, zur Referenz sei ein Blick in das Benutzerhandbuch der HC11IDE bzw. in das Programm selbst angeraten.

5.2.4. Menüs und Hilfsfunktionen - HC_MENU.CPP

Diese Datei enthält Menü- und Statuszeilendefinitionen, sowie einige Hilfsfunktionen. Zu den Menüs und zur Statuszeile ist keine nähere Erläuterung notwendig, an dieser Stelle sollen jedoch einige Erklärungen zu den Hilfsfunktionen nicht fehlen.

Die Funktion ftcmp() (= File Time Compare) vergleicht die Erstellungszeit zweier Dateien. Zuerst wird überprüft, ob eine der beiden Dateien nicht auffindbar ist. Wenn ja, so endet die Funktion mit dem Rückgabewert Null. Sind beide Dateien vorhanden, so wird deren Erstellungsdatum und -zeit verglichen. Ist die Datei mit dem Namen fn1 älter als die Datei mit dem Namen fn2, so wird ein negativer Wert zurückgegeben, ist fn1 jünger als fn2 ein positiver Wert, und sind beide Dateien gleich alt, wird der Wert Null zurückgegeben.

Die global verfügbare Funktion fileexist() überprüft, ob eine Datei mit dem gegebenen Namen existiert. Der Returnwert lautet dementsprechend True oder False.

Schließlich ist in der Quelldatei noch die Funktion path_open() definiert, welche den gesamten DOS PATH (Suchpfad für Programme) absucht, und damit auch Dateien öffnen kann, die ohne explizite Pfadangaben benutzt werden sollen, sich aber irgendwo im Suchpfad befinden. Ansonsten verhält sich diese Funktion bezüglich der Rückgabewerte wie die Funktion fopen() aus der Standardbibliothek.

5.2.5. Serielle Kommunikation - HC_TERM.CPP

Im Quelltext HC_TERM.CPP sind alle Kommunikationselemente des Programms zusammengefaßt, die sich der asynchronen seriellen Schnittstelle bedienen.

Dem Konstruktor der Klasse UTerminal wird die gewünschte Baudrate und der zu verwendende Seriellport mitgeteilt. Diese Parameter werden an die Funktion reinitialize() weitergeleitet.

UTerminal::reinitialize() ermittelt die zur COM-Nummer gehörige Hardwareadresse und initialisiert den UART-Baustein mit dem Protokoll: 8 Bit, keine Parität, 1 Stoppbit. Die Baudrate wird auf den gewünschten bzw. den nächsten erreichbaren Wert gesetzt.

Die Elementfunktion getErrorDesc() (get Error Description) wandelt die im Modul verwendeten Fehlernummern in entsprechende textuelle Fehlermeldungen um. getCOM() und putCOM() lesen bzw. schreiben ein Byte. Es wird im Pollingverfahren gearbeitet. resetTarget() zieht die RTS-Leitung für 2 ms auf Low-Pegel und setzt dadurch die angeschlossene MCU zurück (Anschlußschema siehe vorn). testBreak() prüft, ob über die serielle Schnittstelle ein BREAK (kontinuierliches LOW) empfangen wird. Der HC11 sendet dieses BREAK nach einem Reset, bis die Bootstrap Firmware ein Zeichen empfängt.

Während die bisherigen Elementfunktionen der UTerminal-Klasse einfache Basisfunktionen waren, folgen nun komplexere Routinen. boot() setzt die MCU zurück und sendet ein Zeichen. boot() wird von startRAM() und startEEPROM() benutzt, jeweils mit anderen Start Bytes. loadANDstartRAM() verwendet ebenfalls boot(), sendet jedoch nicht nur das Start Byte (hier: 0xff), sondern weiterhin 256 Byte, die in den HC11-RAM geladen werden. Das vom HC11 zurückgesendete Echo wird auf Identität geprüft. Bleibt die korrekte Rückmeldung aus, wird eine Fehlermeldung erzeugt. Ebenso wird es als Fehler angesehen, wenn die zu transferierende Datei länger ist als die erforderlichen 256 Byte. Ist die Datei jedoch kürzer, wird mit Nullbytes aufgefüllt.

progEEPROM() baut auf der eben beschriebenen Funktion auf, diese Routine lädt erst ein HC11-Programm in den RAM der MCU, und dieses Programm lädt weitere Bytes, die in den EEPROM gebrannt werden. Auch bei der Programmierung wird wieder überprüft, ob das Echo mit dem gesendeten Byte übereinstimmt.

Die Elementfunktion step4step() ist für den Trace Betrieb zuständig. Durch Senden der Startadresse (nur relevant beim ersten Programmschritt) wird die Abarbeitung eines Programmschrittes initiiert. Daraufhin schickt das Target einen Satz Informationsbytes zurück, die über den Prozessorstatus etc. Aufschluß geben. Diese Informationen werden über die Struktur TraceInfo an die aufrufende Funktion zurückgegeben.

5.2.6. Reassemblermodul - HC_REAS.CPP

In diesem Modul wird die Klasse UReAssembler definiert, die einen Reassembler für 68HC11 Code bereitstellt. Die einzige öffentliche Funktion der Klasse ist UReAssembler::reAssemble() am Ende der Datei, die anderen Komponenten werden nur innerhalb der Klasse verwendet.

Der Abschnitt "globals" enthält die Mnemonics der einzelnen HC11 Befehle. Die Anordnung innerhalb der drei Felder mnemo_X entspricht der Lage der Operationen in der Opcode Map der MCU.

UReAssembler::x2y() ist eine Hilfsfunktion zur Änderung eines X-bezogenen Befehls in einen Y-bezogenen Befehl. Dazu wird einfach jedes auftretende Zeichen "X" in "Y" umgewandelt. Wegen des Befehls XGDX und der Tatsache, daß an erster Position niemals eine Umwandlung stattfinden muß, wird das erste Zeichen von dieser Konvertierung ausgeschlossen. Das Verfahren ist sinnvoll aufgrund der internen Behandlung von X- bzw. Y-bezogenen Befehlen, es handelt sich jeweils um den selben Opcode, jedoch kommt bei Y-bezogenen Befehlen noch ein Prebyte hinzu. Siehe dazu auch die HC11 Opcode Map in [3].

Die richtige Assembleranweisung zu einem Opcode (und ggf. einem zugehörigen Prebyte) wird über die Funktion getMnemonic() ermittelt. Bis auf wenige Ausnahmen geschieht dies über die oben definierten mnemo_X Zeichenketten-Arrays. Die Funktion gibt bei einem illegalen Opcode NULL zurück.

Die folgenden Elementfunktionen opRel(), opImm(), opImm16(), opDir(), opExt(), opInd() und opMsk() erzeugen jeweils einen für die angesprochene Adressierungsart (Relativ, Immediate, Immediate 16 Bit, Direct, Extended Memory, Indexed oder Mask) typischen Operandenstring. Ein oder mehrere dieser Strings werden von der Elementfunktion getOperands() zusammengestellt. Zusammen mit dem Mnemonicstring ergibt sich eine komplette reassemblierte Zeile.

Um die Anzahl der zu einem Befehl gehörenden Operandenbytes zu ermitteln, wird die Elementfunktion getByteCnt() verwendet.

Der (Haupt-) Funktion reAssemble() wird der augenblicklich gültige Programcounter (pc), ein Zeiger auf den zu disassemblierenden Code (data) und ein Pufferbereich für die Ablage der reassemblierten Zeile (buf) übergeben. Zuerst wird geprüft, ob es sich um eine Instruktion mit Prebyte handelt. Dann wird aus Prebyte und Opcodebyte der zugehörige Mnemonic ermittelt. Danach wird die Anzahl der Operandenbytes berechnet und die Operandenbytes eingelesen sowie der Funktion getOperands zur Verfügung gestellt. Hat der augenblickliche Befehl das Prebyte 0xcd, so wird der X-bezogene Operandenstring in einen Y-Bezug konvertiert. Ist das Prebyte 0x18 aktiv, wird der ganze String (Mnemonic und Operanden) konvertiert. Schließlich gibt die Funktion die Differenz aus neuem und altem Programcounter, also die Länge des gerade ermittelten Befehls, zurück.

5.2.7. S-Record Konverter - HC_SR2B.CPP

Das Modul nimmt lediglich die Aufgabe der Umwandlung einer Motorola-S-Record- in eine Binärdatei wahr. Es gibt zwar eine Reihe eigenständiger Hilfsprogramme, die diese Funktion realisieren, eine nahtlose Einbindung in den Quelltext erwies sich jedoch als komfortabler, was die Schnelligkeit als auch die Möglichkeit der Fehlererkennung anbelangt.

Es sind nur drei Funktionen implementiert, sie wurden der Applikations-Klasse HC11IDEapp zugeordnet. hexbin() und hexe() sorgen für die Wandlung eines Zeichens bzw. einer ganzen Zeichenkette von der Hexadezimalform in ein ein binäres Datum.

Die Funktion HC11IDEapp::sr2bin() führt die Konvertierung aus. Zu diesem Zweck wird der Funktion der Name der S-Record Datei (objfn), der Name der zu erzeugenden Binärdatei (binfn) und der Adreßbereich, innerhalb dessen gearbeitet werden soll, übergeben. Nachdem beide Dateien geöffnet bzw. erzeugt wurden, werden die Flags cut und empty gesetzt. Ist cut True, so liegen Teile des erzeugten Codes außerhalb des spezifizierten Adreßbereiches. Ist empty True, so ist die erzeugte Binärdatei leer, es liegt also keinerlei Code innerhalb des angegebenen Zielbereiches.

Innerhalb der while-Schleife wird nun Zeile für Zeile der S-Record Datei gelesen und die Codeinformationen extrahiert. Zeilen mit der S-Rocord ID "0" werden übersprungen, ID "1" führt zum Einlesen und ID "9" führt zum Abbruch (letzte Zeile). Andere ID´s sind nicht zulässig.

Eine Fehlermeldung wird erzeugt, wenn die in der S-Record Datei enthaltenen Code-Bereiche nicht in aufsteigender Reihenfolge vorliegen. Leere Bereiche werden mit 0xff aufgefüllt.

Die letzte Zeile kann mit der ID "9" gekennzeichnet sein, eine solche Zeile muß aber nicht unbedingt vorhanden sein. Bei Erreichen des Dateiendes wird die Konvertierung ebenfalls ordnungsgemäß abgeschlossen.

Nach Schließen der Dateien wird eine Fehlermeldung erzeugt, wenn die Binärdatei leer geblieben ist, oder es wird eine Warnung zurückgegeben, wenn die Ausgabe abgeschnitten wurde. Das Zeichen "\t" zu Beginn der Warnung ist ein Flag für die aufrufende Funktion, daß es sich nicht um einen fatalen Fehler handelt, der zum Abbruch führt, sondern nur eine Warnungsmitteilung erzeugt werden soll, bevor die Abarbeitung fortgesetzt wird.

Erfolgte die Konvertierung ohne irgendwelche Probleme, wird der Wert NULL zurückgegeben.

5.2.8. Heap Information - HC_HEAP.CPP

Dieses Modul wurde aus den Turbo Vision Demo Support Files extrahiert. Es stellt eine Anzeige des verfügbaren Hauptspeicherplatzes zur Verfügung. Diese Anzeige hat lediglich den Stellenwert einer Zusatzinformation bei der Programmanwendung. Sie wurde vorrangig implementiert, um bei der Entwicklung auftretende Probleme mit der Speicherverwaltung aufzudecken.

5.2.9. modifizierte FileViewer Klasse - FILEVIEW.CPP und .H

Die Datei FILEVIEW.CPP wurde komplett aus den Turbo Vision Demo Support Files übernommen und dahingehend modifiziert, daß sämtliche define- und include-Anweisungen entfernt und durch die Includedatei HC11IDE.H ersetzt wurden. In der Includedatei HC11IDE.H ist nun auch die TV Includedatei FILEVIEW.H integriert.

5.2.10. Hilfsprogramm ERR_FIND.C

In das System wurde der Motorola Freeware Assembler ASMHC11 eingebunden. Leider ist dieser Assembler nicht dahingehend vorbereitet, auftretende Fehler an eine aufrufende Stapelverarbeitungsdatei weiterzumelden. Aus diesem Grund wurde das Utility ERR_FIND erforderlich. Dieses Utility durchsucht die vom Assembler erzeugte Listingdatei nach dem ersten Fehler. Ein Fehlerzeile wird identifiziert durch das Pattern "** ERROR" in dieser Zeile, wohingegen eine normale Codezeile stets mit einer Ziffer beginnt. Eingestreute Seitenvorschübe werden ausgeblendet und es wird eine Datei angelegt mit zwei Zeilen: Der betroffenen Codezeile und der erzeugten Fehlerzeile. Ist die Fehlerdatei leer, bedeutet das, daß kein Fehler aufgetreten ist.

Der Quelltext ERR_FIND.C wird separat übersetzt und als ERR_FIND.EXE in der Stapelverarbeitungsdatei A!.BAT zur Auffindung von Assemblerfehlern verwendet.

5.2.11. Support Files A!.BAT und S!.BAT

Diese beiden Files sind Stapelverarbeitungsdateien zur Einbindung der externen Programme Assembler (ASMHC11) und Simulator (TESTE68). Sie bekommen beide als erstes Argument den gerade bearbeiteten Dateinamen (ohne Endung) übergeben. Zusätzlich wird A!.BAT als zweites Argument der Wert der Assemblerkonstanten PROG (Programm-Anfangsadresse) in Form eines 4-stelligen Hexadezimalwertes übergeben.

6. Ausblick

Die hier vorgestellte Integrierte Entwicklungsumgebung ist einerseits universell für die Entwicklung von HC11 Software einsetzbar, andererseits ist sie aber auch in vielerlei Hinsicht zugeschnitten auf die Belange des geplanten Einsatzes in einem Mikrorechentechnik-Labor. Somit ergeben sich noch vielerlei Ansatzpunkte zur Erweiterung und Verbesserung der Software in Hinblick auf andere Einsatzfälle oder spezielle Hardwareerfordernisse.

Bei einer Fortführung der Arbeiten an der Software könnten folgende Ideen als nächstes aufgegriffen werden:

- Einbeziehung externer Speicher-Ressourcen
- Berücksichtigung weiterer MCU-Typen aus der HC11 Familie
- Aufnahme einer Terminal-Emulation
- freie Konfigurierbarkeit der seriellen Schnittstellen
- Verwendung einer Konfigurationsdatei
- Implementation eines eigenen Cross-Assembler Moduls, unter anderem zur Verbesserung der Möglichkeiten zur Fehlerbehandlung des Assemblers

Bereits während der Entwicklungsarbeiten an der Soft- und Hardware der Praktikumsplätze zeichnete sich ab, daß auch andere Lehreinrichtungen Interesse an einer Übernahme der gewonnenen Erkenntnisse und Arbeitsergebnisse haben könnten. Erste Absprachen dazu gab es bereits mit mehreren Fachhochschulen in der Bundesrepublik. Ein Transfer von Know-How bzw. Leistungen könnte sich einerseits auf die Software beschränken (die HC11IDE hat Freeware Status, die HTWK Leipzig kann darüber frei verfügen), oder aber auch durch Hardware ergänzt werden. An dieser Stelle sei auf die Belegarbeiten der Herren Haase und Gatzke verwiesen, die, unter der Federführung des Autors dieser Arbeit, ein Konzept für die Hardwarebasis im MRT-Labor erarbeiteten, dieses innerhalb kürzester Zeit praktisch realisierten und für die Nutzung im Praktikumsbetrieb einrichteten.

7. Zusammenfassung

Ausgehend von der Aufgabenstellung, die komplette Hard- und Software zur Errichtung des neuen Mikrorechentechnik Labors der HTWK Leipzig (FH) zu schaffen, wurden Überlegungen über einen möglichen Lösungsweg angestellt. Dieser Lösungsweg sollte zwar kostengünstig sein, dennoch aber auch komfortable Entwicklungshilfsmittel zur Verfügung stellen. Das gesamte Projekt wurde aufgeteilt in die Soft- und Hardwarerealisierung. Für die Hardware fanden einige preiswerte industrielle Controllermodule Verwendung, die durch universelle Peripheriebaugruppen und mechanische Trägerkomponenten aus eigener Entwicklung zu einem Baukastensystem komplettiert wurden.

Nach Betrachtung der sich bietenden Lösungsansätze zur Realisierung der Entwicklungssoftware wurden vorhandene Programme aus Kosten- bzw. Performancegründen verworfen und ein eigenes Konzept erstellt. Das Look-And-Feel der realisierten Software ist nahezu jedem Benutzer durch die Turbo-... Produkte der Firma Borland bekannt. Bereits vor Vorhandensein der Dokumentation wurde die Software im Labor von vielen Studenten umfangreich genutzt. Das allein zeigt bereits, daß die Prämissen "Anwenderakzeptanz" und "intuitive Handhabbarkeit" sehr gut umgesetzt werden konnte.

Die gefundene Lösung zur Implemetation eines Singlestep-Betriebes ist in dieser Art und Weise neu und beweist, daß zumindest im Gebiet der HC11 Mikrocontroller nicht unbedingt kostenintensive Hardwareemulatoren oder umfangreich ausgebaute - und damit unübersichtliche - MCU-Evaluationboards eingesetzt werden müssen, um Software komfortabel austesten zu können. Die gestellten Forderungen in Bezug auf Funktionalität und minimale Ressourcenbelegung konnten voll erfüllt werden.

Das Resultat in Form der eingerichteten Praktikumsplätze und der jedermann zugänglichen Entwicklungssoftware hat sich in der Praxis bereits bestens bewährt. Die Beachtung, die das Projekt bisher innerhalb und außerhalb der Hochschule fand, insbesondere das wohlwollende Interesse des GB Halbleiter der Firma Motorola Deutschland sowie weiterer Unternehmen der freien Wirtschaft sind höchste Anerkennung für die bisherige Arbeit des Autors und aller Personen, die sich in das Projekt eingebracht haben.

8. Quellennachweis/Literaturverzeichnis

[1] Thamm, Oliver: Der Microcontroller 68HC11. radio fernsehen elektronik H. 1/1993, S. 20ff. Verlag Technik, Berlin 1993

[2] Thamm, Oliver: Miniaturcomputer Zwerg11A. radio fernsehen elektronik H. 2/1993, S. 40ff. Verlag Technik, Berlin 1993

[3] Motorola Inc.: MC68HC811E2 Technical Data. Part No. MC68HC811E2/D. Motorola Inc. 1991

[4] Motorola Inc.: HC11 Reference Manual. Part No. M68HC11RM/AD REV1. Motorola Inc. 1990

[5] Graf, Rudolf: TESTE68 - Befehlsinterpreter für die Mikrocontroller der M68HC11-Familie. ROHDE Interface, Poing 1992

[6] Klemm, Frank und Thamm, Oliver: Z80mini3 Monitor V4, Debug-Monitor incl. Singlestep-Betrieb für den Einplatinenrechner Z80mini3. MCT GbR, Leipzig 1993

[7] Paul, Hennes; Scherer, Erhard und Scherer, Walter: Zwer11A und Zwerg11plus - MC68HC11 Einplatinenrechner - Hardware Handbuch. MCT Paul & Scherer Mikrocomputertechnik GmbH, Berlin 1992

[8] Motorola Inc.: HCMOS Single-Chip Microcontroller MC68HC11A8 - Advance Information. Part No. MC68HC11A8/D. Motorola Inc. 1988

[9] Borland GmbH: Turbo Vision für C++ - Programmierhandbuch. Art.-Nr. 138301. Borland GmbH, Starnberg 1991

[10] Borland GmbH: Turbo Vision für C++ - Referenzhandbuch. Art.-Nr. 138300. Borland GmbH, Starnberg 1991

[11] Belegarbeit des Herrn Steffen Haase über Hardwarekomponenten und Praktikumsaufgaben für das Mikrorechentechnik Labor, HTWK Leipzig (FH) 1994

[12] Belegarbeit des Herrn Ingo Gatzke über Hardwarekomponenten und Praktikumsaufgaben für das Mikrorechentechnik Labor, HTWK Leipzig (FH) 1994

Anhang A: Benutzerhandbuch HC11IDE

- liegt separat bei -

Anhang B: EEPROMer Programm für den HC11

```
**============================================================================**
**                                                                            **
**  Titel:    EEPROG.ASM                                                      **
**  Funktion: Programmiert den internen EEPROM des HC11. Übergabe der zu      **
**            programmierenden Bytes über das SCI (7812 Baud). Die ersten     **
**            beiden Bytes bestimmen die Anfangsadresse. Die folgenden        **
**            werden programmiert und das Ergebnis als Echo zurückgesendet.   **
**  Version:  1.2                                                             **
**  Datum:    19.02.94                                                        **
**  Autor:    Oliver Thamm                                                    **
**                                                                            **
**============================================================================**

**--- include files ----------------------------------------------------------**

        include ..\lib\hc11.h

**--- main -------------------------------------------------------------------**

        org $0000

        clr     obprot,x                /* disable block protect (E2) */
        bsr     sciget                  /* get starting address */
        tab                             /* L-byte */
        bsr     sciget                  /* H-byte */
        xgdx
        ldab    #$16                    /* ID for EEPROM write */

loop    bsr     sciget                  /* get byte from sci */
        bsr     eewrite                 /* write to EEPROM */
        bsr     sciput                  /* return EEPROM data */
        inx                             /* next address... */
        bra     loop


**--- eeprom write -----------------------------------------------------------**

*       ACCA = data
*       ACCB = $06 for config or $16 for EEPROM
*       IX   = Address
*       return ACCA = EEPROM byte

eewrite cmpa    0,x                     /* check destination */
        beq     eewend                  /* nothing to write! */
        stab    pprog                   /* set byte erase mode */
        staa    0,x                     /* dummy write */
        inc     pprog                   /* erase voltage on */
        pshb
        psha
        bsr     delay10                 /* delay 10 ms */
        ldaa    #$02
        staa    pprog                   /* programming mode */
        pula
        staa    0,x                     /* write data */
        inc     pprog                   /* prog. voltage on */
```

```
        bsr     delay10                 /* delay 10 ms */
        clr     pprog                   /* normal read mode */
        ldaa    0,x                     /* return EEPROM byte */
        pulb
eewend  rts


**--- 10 ms delay ------------------------------------------------------**

delay10 ldd     #20000
        addd    tcnt
dly_wt  cpd     tcnt                    /* 500ns delay steps */
        bpl     dly_wt
        rts


**--- get char from sci to ACCA ----------------------------------------**

sciget  ldaa      scsr
        anda      #$20                  /* receiver buffer full? */
        beq       sciget                /* no, try again... */
        ldaa      scdr                  /* read from buffer */
        rts


**--- put char in ACCA to sci ------------------------------------------**

sciput  tst       scsr                  /* transmitter buffer empty? */
        bpl       sciput                /* no, try again... */
        staa      scdr                  /* write to buffer */
        rts


**--- fill 256 bytes ---------------------------------------------------**

        org $00ff

        nop


**--- fine -------------------------------------------------------------**

        end
```

Anhang C: Trace-Programm für den HC11

```
**============================================================================**
**                                                                            **
**  Titel:    TRACE.ASM                                                       **
**  Funktion: Abarbeitung eines Anwender-Programms im Einzelschrittbetrieb    **
**  Version:  1.4                                                             **
**  Datum:    19.02.94                                                        **
**  Autor:    Oliver Thamm                                                    **
**                                                                            **
**============================================================================**

**--- include files ----------------------------------------------------------**

          include   ..\lib\hc11.h         /* Equates etc. */


**--- startup code -----------------------------------------------------------**

          org $0000

start     bsr       sciget                /* get starting address */
          tab                             /* L-byte */
          bsr       sciget                /* H-byte */
          xgdy                            /* IY contains starting address */

prep      bset      ohprio,x,#%00001111   /* give highest priority to OC5 */
          clr       otmsk2,x              /* set timer prescaler */
          ldd       otcnt,x     /* 1/01   get system counter */
          addd      #23         /* 4/05   add ticks until jmp instruction */
          std       otoc5,x     /* 5/10   set system counter */
          ldaa      #$08        /* 2/12   output capture 5 mask */
          staa      otmsk1,x    /* 4/16   enable OC5 */
          staa      otflg1,x    /* 4/20   clear pending OC5 Int's */
          cli                   /* 2/22   global Int enable */
          jmp       0,y         /* 1/23   jump to user program */


**--- misc. I/O-routines -----------------------------------------------------**

          org $0070


**--- get char from sci to ACCA ----------------------------------------------**

sciget    ldaa      scsr
          anda      #$20                  /* receiver buffer full? */
          beq       sciget                /* no, try again... */
          ldaa      scdr                  /* read from buffer */
          rts


**--- put char in ACCA to sci ------------------------------------------------**

sciput    tst       scsr                  /* transmitter buffer empty? */
          bpl       sciput                /* no, try again... */
          staa      scdr                  /* write to buffer */
```

```
          rts


**--- Interrupt Service Routine ----------------------------------------**

isr       tsx                           /* IX now contains SP + 1  */
          ldy       7,x                 /* IY now contains user PC */
          ldab      #9
regread   ldaa      0,x                 /* read out CCR,ACCB,ACCA, */
          bsr       sciput              /* IXH,IXL,IYH,IYL,PCH,PCL */
          ldaa      0,y                 /* read out code bytes     */
          bsr       sciput              /* <PC>,<PC+1>,...,<PC+8>  */
          inx
          iny
          decb
          bne       regread

          dex                           /* read out SP */
          xgdx                          /* (value before interrupt) */
          bsr       sciput
          tba
          bsr       sciput

          ldx       #$1000              /* read out port A,B,C,D,E */
          ldaa      oporta,x
          bsr       sciput
          ldaa      oportb,x
          bsr       sciput
          ldaa      oportc,x
          bsr       sciput
          ldaa      oportd,x
          bsr       sciput
          ldaa      oporte,x
          bsr       sciput
done_     ldaa      #$0A                /* LineFeed */
          bsr       sciput

          bsr       sciget              /* compatibility to startup code */
          bsr       sciget              /* next step, please! */
prep_     bset      ohprio,x,#%00001111 /* give highest priority to OC5 */
          clr       otmsk2,x            /* set timer prescaler */
          ldd       otcnt,x     /* 1/01   get system counter */
          addd      #33         /* 4/05   add ticks until 1st user instr. */
          std       otoc5,x     /* 5/10   set system counter */
          ldaa      #$08        /* 2/12   output capture 5 mask */
          staa      otmsk1,x    /* 4/16   enable OC5 */
          staa      otflg1,x    /* 4/20   clear pending OC5 Int's */
          rti                   /*12/32   return to user prg */
          *                     /* 1/33   1st user instruction */

**--- output compare 5 pseudo-vector -----------------------------------**

          org $00D3

OC5VECT   jmp       isr


**--- fill 256 bytes ---------------------------------------------------**
```

```
          org $00ff

          nop

**--- fine ----------------------------------------------------------------**

        end
```

Anhang D: PC-Software (Sourcecodes)

D.1. HC11IDE.H

```
/*=========================================================================*/
/*                                                                         */
/*  Titel:    HC11IDE.H                                                    */
/*  Funktion: Prototypes for all user defined classes, structures etc.     */
/*  Version:  1.00                                                         */
/*  Datum:    22.2.94                                                      */
/*  Autor:    Oliver Thamm                                                 */
/*                                                                         */
/*=========================================================================*/


/*-- defines ------------------------------------------------------------*/

#define Uses_TApplication
#define Uses_TButton
#define Uses_TChDirDialog
#define Uses_TCheckBoxes
#define Uses_TCollection
#define Uses_TDeskTop
#define Uses_TDialog
#define Uses_TDrawBuffer
#define Uses_TEditor
#define Uses_TEditWindow
#define Uses_TFileDialog
#define Uses_TFileEditor
#define Uses_THistory
#define Uses_TInputLine
#define Uses_TKeys
#define Uses_TLabel
#define Uses_TMenuBar
#define Uses_TMenuItem
#define Uses_TObject
#define Uses_TPoint
#define Uses_TProgram
#define Uses_TRadioButtons
#define Uses_TRect
#define Uses_TScroller
#define Uses_TSItem
#define Uses_TStatusDef
#define Uses_TStatusItem
#define Uses_TStatusLine
#define Uses_TStreamableClass
#define Uses_TSubMenu
#define Uses_TView
#define Uses_TWindow
#define Uses_MsgBox


/*-- includes -----------------------------------------------------------*/

#include <tv.h>

#include <stdio.h>
```

```
#include <stdlib.h>
#include <stdarg.h>
#include <strstrea.h>
#include <iomanip.h>
#include <dos.h>
#include <bios.h>
#include <dir.h>
#include <alloc.h>
#include <time.h>
#include <ctype.h>
#include <string.h>
#include <fstream.h>

#include "fileview.h"


/*-- more defines ------------------------------------------------------*/

#define _COM_TxBufEmpty     0x2000
#define _COM_RxDataAvail    0x0100
#define _COM_BrkDetect      0x1000

#define edResetFailed       500
#define edTxTimeout         501
#define edDlFileNotFnd      502
#define edEOF               503
#define edHndshakeFailed    504
#define edRamFile2Long      505
#define edPgmError          506
#define edEepFile2Long      507


/*-- constants ---------------------------------------------------------*/

const
  cmOpen            = 100,
  cmNew             = 101,
  cmChangeDrct      = 102,
  cmDosShell        = 103,
  cmCalculator      = 104,
  cmShowClip        = 105,
  cmAssemble        = 106,
  cmRebuild         = 107,
  cmExecute         = 108,
  cmWhoIsThere      = 109,
  cmTrace           = 110,
  cmTargetopts      = 111,
  cmOptionsModified = 112,
  cmAboutCmd        = 113,
  cmSimulate        = 114,
  cmTraceFrom       = 115,
  cmUpdateCommands  = 116;


/*-- forward definitions -----------------------------------------------*/

class  UEditWindow;
class  THeapView;
```

```
struct UTraceInfo;


/*-- externals ------------------------------------------------------------*/

extern TEditWindow *clipWindow;


/*-- function prototypes --------------------------------------------------*/

TDialog *createFindDialog();
TDialog *createReplaceDialog();
TWindow *createInfoWin(const char *);
TDialog *createInfoDlg(const char *);
TDialog *createTraceDlg();
TDialog *createTargOptDlg();
TDialog *createAboutDlg();
TDialog *createTraceFromDlg();
ushort execDialog( TDialog *d, void *data );
ushort doEditDialog( int dialog, ... );
int ftcmp(const char *, const char *);
Boolean fileexist(const char *);
FILE * path_fopen(const char *fn, const char *mode);


/*-- class UTerminal ------------------------------------------------------*/

class UTerminal {

public:
    UTerminal(unsigned BaudRate, unsigned ComPort);
    void reinitialize(unsigned BaudRate, unsigned ComPort);
    int getCOM();
    Boolean putCOM(char c);
    Boolean resetTarget();
    Boolean startRAM();
    Boolean startEEPROM();
    Boolean loadANDstartRAM(const char *filename);
    Boolean progEEPROM(const char *filename, unsigned eeprom_size, unsigned
eeprom_start);
    const char *getErrorDesc();
    Boolean step4step(UTraceInfo *);

private:
    unsigned theComPort;
    unsigned thePortBase;
    int errordesc;
    Boolean testBreak();
    Boolean boot(unsigned char c);
    };


/*-- class UTargOpt -------------------------------------------------------*/

class UTargOpt {

public:
    ushort MCUtype;
```

```
    ushort Loadprg;
    };


/*-- class HC11IDEapp ----------------------------------------------------*/

class HC11IDEapp : public TApplication {

public:
    HC11IDEapp();
    virtual void handleEvent( TEvent& event );
    static TMenuBar *initMenuBar( TRect );
    static TStatusLine *initStatusLine( TRect );
    virtual void outOfMemory();

private:
    UEditWindow *openEditor( const char *fileName, Boolean visible );
    void fileOpen();
    void fileNew();
    void changeDir();
    void dosShell();
    void showClip();
    void tile();
    void cascade();
    const char * make(ushort);
    void targetOpts();
    UTargOpt TargetOptions;
    const char * sr2bin(const char *, const char *, unsigned, unsigned);
    unsigned hexbin(char c);
    unsigned hexe(char *s, unsigned sl);
    TWindow *StatusWindow;
    void trace();
    void idle();
    THeapView *heap;
    void about();
    };


/*-- class UEditWindow ---------------------------------------------------*/

class UEditWindow : public TEditWindow {

public:
    UEditWindow(const TRect&, const char *, int);
    void handleEvent(TEvent&);
private:
    void shutDown();
    void updateCommands();
    };


/*-- class UErrViewer ----------------------------------------------------*/

class UErrViewer : public TWindow {

public:
    UErrViewer(TRect, const char *);
    void close();
```

```
    void handleEvent(TEvent&);
    };


/*-- struct fn_components ------------------------------------------------*/

struct fn_components {
    char drive[MAXDRIVE];
    char dir[MAXDIR];
    char name[MAXFILE];
    char ext[MAXEXT];
    };


/*-- struct UTraceInfo ---------------------------------------------------*/

struct UTraceInfo {
    unsigned CCR;
    unsigned ACCB, ACCA;
    unsigned IXH, IXL, IYH, IYL;
    unsigned PCH, PCL, SPH, SPL;
    unsigned char CODE[9];
    unsigned PORTA, PORTB, PORTC, PORTD, PORTE;
    };


/*-- class UTraceView ----------------------------------------------------*/

class UTraceView : public TView {

private:
    UTraceInfo traceInfo;

public:
    UTraceView(TRect &);
    void draw();
    void handleEvent(TEvent &);
    void update();
    };


/*-- class UReAssembler --------------------------------------------------*/

class UReAssembler {

private:
    int x2y(char *s);
    char * getMnemonic(unsigned char opcode, unsigned char& prefix);
    char * opRel(unsigned pc, unsigned offset);
    char * opImm(unsigned value);
    char * opImm16(unsigned value);
    char * opDir(unsigned address);
    char * opExt(unsigned address);
    char * opInd(unsigned index);
    char * opMsk(unsigned value);
    char * getOperands(unsigned char opcode, unsigned long loperand, unsigned pc);
    unsigned getByteCnt(unsigned char opcode);
public:
```

```
    unsigned reAssemble(unsigned pc, unsigned char *data, char * buffer);
    };


/*-- class THeapView ------------------------------------------------------*/

class THeapView : public TView {

public:
    THeapView( TRect& r );
    virtual void update();
    virtual void draw();
    virtual long heapSize();
private:
    long oldMem, newMem;
    char heapStr[16];
    };


/*-- fine -----------------------------------------------------------------*/
```

D.2. HC_MAIN.CPP

```
/*=========================================================================*/
/*                                                                         */
/*  Titel:    HC_MAIN.CPP                                                  */
/*  Funktion: Hauptmodul, enthält neben main() und handleEvent() die Imple- */
/*            mentation der Klassen HC11IDEapp (Nachfahre von TApplication) */
/*            sowie UErrViewer, UTraceView und UEditWindow                 */
/*  Version:  1.00                                                         */
/*  Datum:    22.2.94                                                      */
/*  Autor:    Oliver Thamm                                                 */
/*                                                                         */
/*=========================================================================*/


/*-- defines ----------------------------------------------------------*/


/*-- includes ---------------------------------------------------------*/

#include "hc11ide.h"


/*-- globals ----------------------------------------------------------*/

TEditWindow *clipWindow;
TRect rErrWin;
UErrViewer *errWindow;
UTerminal *ter;
UReAssembler reAss;
unsigned PrgStart, PrgEnd, PrgSize;
unsigned menuState;


/*-- UErrViewer constructor -------------------------------------------*/

UErrViewer::UErrViewer(TRect r, const char *fileName) :
    TWindow(r , fileName, wnNoNumber),
    TWindowInit(&UErrViewer::initFrame) {

    options &= ~ofTileable;
    TRect r1( getExtent() );
    r1.grow(-1, -1);
    insert(new TFileViewer( r1,
                            standardScrollBar(sbHorizontal | sbHandleKeyboard),
                            standardScrollBar(sbVertical | sbHandleKeyboard),
                            fileName) );
    }


/*-- UErrViewer::handleEvent() ----------------------------------------*/

void UErrViewer::handleEvent(TEvent& event) {


    if(event.what == evKeyDown)
        switch(event.keyDown.keyCode) {
```

```
            case kbEsc:
                //message(this,evCommand,cmClose,0);
                //clearEvent(event);
                event.what = evCommand;
                event.message.command = cmClose;
                break;
            }
    TWindow::handleEvent(event);
    }


/*-- UErrViewer::close() ------------------------------------------------*/

void UErrViewer::close() {

    rErrWin = getBounds();
    errWindow = NULL;
    TWindow::close();
    }


/*-- UTraceView constructor ---------------------------------------------*/

UTraceView::UTraceView(TRect &r) : TView(r) {

    update();
    }


/*-- ito2x() ------------------------------------------------------------*/

char * ito2x(int i, char *buf) {

    int j = ((i>>4) & 0x0f) + '0';
    if(j > '9') j += 7;
    *buf = j;
    j = (i & 0x0f) + '0';
    if(j > '9') j += 7;
    *(buf+1) = j;
    *(buf+2) = '\0';
    return(buf);
    }


/*-- ito8b() ------------------------------------------------------------*/

char * ito8b(int i, char *buf) {

    unsigned mask;
    char *buf1 = buf;

    mask=0x80;
    while(mask) {
        *buf1++ = (i & mask) ? '1' : '0';
        mask >>= 1;
        }
    *buf1 = '\0';
    return(buf);
```

```
    }


/*-- UTraceView::draw() ------------------------------------------------*/

void UTraceView::draw() {

    char str[40];
    TDrawBuffer buf;
    char b[40];

    buf.moveChar(0,' ',getColor(6),size.x);
    writeLine(0,0,size.x,size.y,buf);

    writeStr( 2, 1, "ACCA:"     ,6);
    writeStr( 2, 2, "ACCB:"     ,6);
    writeStr( 2, 3, "IX:"       ,6);
    writeStr( 2, 4, "IY:"       ,6);
    writeStr( 2, 6, "CCR:"      ,6);
    writeStr( 2, 7, "SXHINZVC"  ,16);
    writeStr(13, 1, "PC:"       ,6);
    writeStr(23, 1, "SP:"       ,6);
    writeStr(23, 3, "76543210"  ,6);
    writeStr(16, 4, "PORTA:"    ,6);
    writeStr(16, 5, "PORTB:"    ,6);
    writeStr(16, 6, "PORTC:"    ,6);
    writeStr(16, 7, "PORTD:"    ,6);
    writeStr(16, 8, "PORTE:"    ,6);
    writeStr( 2,10, "Instruction:",6);

    writeStr( 8, 1, ito2x(traceInfo.ACCA, str), 16);
    writeStr( 8, 2, ito2x(traceInfo.ACCB, str), 16);
    writeStr( 6, 3, ito2x(traceInfo.IXH, str), 16);
    writeStr( 8, 3, ito2x(traceInfo.IXL, str), 16);
    writeStr( 6, 4, ito2x(traceInfo.IYH, str), 16);
    writeStr( 8, 4, ito2x(traceInfo.IYL, str), 16);
    writeStr(17, 1, ito2x(traceInfo.PCH, str), 16);
    writeStr(19, 1, ito2x(traceInfo.PCL, str), 16);
    writeStr(27, 1, ito2x(traceInfo.SPH, str), 16);
    writeStr(29, 1, ito2x(traceInfo.SPL, str), 16);
    writeStr( 2, 8, ito8b(traceInfo.CCR,   str), 16);
    writeStr(23, 4, ito8b(traceInfo.PORTA, str), 16);
    writeStr(23, 5, ito8b(traceInfo.PORTB, str), 16);
    writeStr(23, 6, ito8b(traceInfo.PORTC, str), 16);
    writeStr(23, 7, ito8b(traceInfo.PORTD, str), 16);
    writeStr(23, 8, ito8b(traceInfo.PORTE, str), 16);

    unsigned pc = (traceInfo.PCH << 8) | traceInfo.PCL;
    unsigned valid = reAss.reAssemble(pc, traceInfo.CODE, b);
    sprintf(str,"%-29s", (valid ? b : "Invalid") );
    writeStr( 2,11, str, 16);
    }


/*-- UTRACEVIEW::handleEvent() -----------------------------------------*/

void UTraceView::handleEvent(TEvent& event) {
```

```
    TView::handleEvent(event);

    if(event.what == evCommand)
        switch(event.message.command) {
            case cmTrace:
                update();
                clearEvent(event);
                break;
            }
    }


/*-- UTraceView::update() ------------------------------------------------*/

void UTraceView::update() {

    if(ter->step4step(&traceInfo) == False) {
        execDialog(createInfoDlg("TraceInfo not received"),0);
        }
    drawView();
    }


/*-- UEditWindow constructor ---------------------------------------------*/

UEditWindow::UEditWindow(const TRect& r, const char *fileName, int aNumber) :
    TEditWindow(r, fileName, aNumber),
    TWindowInit(&UEditWindow::initFrame) {

    };


/*-- UEditWindow::handleEvent() ------------------------------------------*/

void UEditWindow::handleEvent(TEvent& event) {

    if(event.what == evCommand)
        switch(event.message.command) {
            case cmWhoIsThere:
                clearEvent(event);
                break;
            case cmOptionsModified:
                editor->modified = True;
                clearEvent(event);
                break;
            case cmUpdateCommands:
                updateCommands();
                clearEvent(event);
                break;
            }
    TEditWindow::handleEvent(event);
    }


/*-- UEditWindow::shutDown() ---------------------------------------------*/

void UEditWindow::shutDown() {
    TCommandSet ts;
```

```
    ts += cmAssemble;
    ts += cmRebuild;
    ts += cmExecute;
    ts += cmSimulate;
    ts += cmFind;
    ts += cmReplace;
    ts += cmSearchAgain;
    disableCommands( ts );
    TEditWindow::shutDown();
    }


/*-- UEditWindow::updateCommands() ------------------------------------------*/

void UEditWindow::updateCommands() {
    TCommandSet ts;
    ts += cmAssemble;
    ts += cmRebuild;
    ts += cmExecute;
    ts += cmSimulate;
    ts += cmFind;
    ts += cmReplace;
    ts += cmSearchAgain;
    enableCommands( ts );
    }


/*-- HC11IDEapp::openEditor() -----------------------------------------------*/

UEditWindow *HC11IDEapp::openEditor( const char *fileName, Boolean visible )
{
    TRect r = deskTop->getExtent();
    TView *p = validView( new UEditWindow( r, fileName, wnNoNumber ) );
    if( !visible )
        p->hide();
    deskTop->insert( p );
    TCommandSet ts;
    ts += cmAssemble;
    ts += cmRebuild;
    ts += cmExecute;
    ts += cmSimulate;
    enableCommands( ts );
    return (UEditWindow *)p;
}


/*-- HC11IDEapp constructor -------------------------------------------------*/

HC11IDEapp::HC11IDEapp() :
    TProgInit( HC11IDEapp::initStatusLine,
               HC11IDEapp::initMenuBar,
               HC11IDEapp::initDeskTop
             ),
    TApplication() {

    TRect r;

    TCommandSet ts;
```

```
    ts += cmSave;
    ts += cmSaveAs;
    ts += cmCut;
    ts += cmCopy;
    ts += cmPaste;
    ts += cmClear;
    ts += cmUndo;
    ts += cmFind;
    ts += cmReplace;
    ts += cmSearchAgain;
    ts += cmAssemble;
    ts += cmRebuild;
    ts += cmExecute;
    ts += cmSimulate;
    disableCommands( ts );

    TEditor::editorDialog = doEditDialog;

    r = getExtent();                            // Create the heap view.
    r.a.x = r.b.x - 13;     r.a.y = r.b.y - 1;
    heap = new THeapView(r);
    insert(heap);

    r = deskTop->getExtent();
    clipWindow = (TEditWindow *) validView(new TEditWindow(r, 0, wnNoNumber));
    clipWindow->hide();
    deskTop->insert(clipWindow);
    TEditor::clipboard = clipWindow->editor;
    TEditor::clipboard->canUndo = False;

    rErrWin = r;
    rErrWin.a.y = rErrWin.b.y - 5;
    errWindow = NULL;
    StatusWindow = NULL;
    TargetOptions.MCUtype = 0;
    TargetOptions.Loadprg = 1;
    PrgStart = 0xb600;
    PrgEnd   = 0xb7ff;
    PrgSize  = 0x0200;
    ter = new UTerminal(7812,1);
    }


/*-- HC11IDEapp::idle() ------------------------------------------------*/

void HC11IDEapp::idle() {

    TProgram::idle();
    heap->update();
    message(this,evCommand,cmUpdateCommands,0);
    }


/*-- HC11IDEapp::fileOpen() --------------------------------------------*/

void HC11IDEapp::fileOpen() {

    char openfileName[MAXPATH];
```

```
    strcpy(openfileName,"*.ASM");
    if( execDialog( new TFileDialog( "*.ASM", "Open file",
        "~N~ame", fdOpenButton, 100 ), openfileName) != cmCancel )
        {
        openEditor(openfileName,True);
        }
    }


/*-- HC11IDEapp::fileNew() ------------------------------------------------*/

void HC11IDEapp::fileNew() {

    openEditor(0,True);
    }


/*-- HC11IDEapp::changeDir() ----------------------------------------------*/

void HC11IDEapp::changeDir() {

    execDialog( new TChDirDialog( cdNormal, 0 ), 0 );
    }


/*-- HC11IDEapp::dosShell() -----------------------------------------------*/

void HC11IDEapp::dosShell() {

    suspend();
    system("cls");
    cout << "Type EXIT to return...";
    system( getenv( "COMSPEC"));
    resume();
    redraw();
    }


/*-- HC11IDEapp::showClip() -----------------------------------------------*/

void HC11IDEapp::showClip() {

    clipWindow->select();
    clipWindow->show();
    }


/*-- HC11IDEapp::tile() ---------------------------------------------------*/

void HC11IDEapp::tile()
{
    deskTop->tile( deskTop->getExtent() );
}


/*-- HC11IDEapp::cascade() ------------------------------------------------*/
```

```
void HC11IDEapp::cascade()
{
    deskTop->cascade( deskTop->getExtent() );
}


/*-- HC11IDEapp::make() ------------------------------------------------------*/

const char * HC11IDEapp::make(ushort cmd) {

    const char * currentSRC;
    char currentFN[MAXPATH];
    char currentOBJ[MAXPATH];
    char currentERR[MAXPATH];
    char currentBIN[MAXPATH];
    char doscmd[120];
    fn_components fn;
    UEditWindow * editWinOnTop;
    TFileEditor * edi;
    const char * resultInfo;

    // fokussiertes Editorfenster ermitteln und sicherstellen, daß
    // der aktuelle Inhalt als File auf der Platte zu finden ist:
    editWinOnTop = (UEditWindow *)message(this,evCommand,cmWhoIsThere,NULL);
    if(!editWinOnTop) {
        return("Nothing to do!");
        }
    edi = editWinOnTop->editor;
    if(edi->modified || edi->fileName==EOS) {
        if(edi->save() == False)
            return("Save file before use");
        }

    StatusWindow = createInfoWin("Assembling, please wait...");
    deskTop->insert(StatusWindow);

    // Dateiname des Editorfensters ermitteln und die Namen der
    // daraus abgeleiteten Dateien bilden
    currentSRC = edi->fileName;
    fnsplit(currentSRC, fn.drive, fn.dir, fn.name, fn.ext);
    fnmerge(currentOBJ, fn.drive, fn.dir, fn.name, ".OBJ");
    fnmerge(currentERR, fn.drive, fn.dir, fn.name, ".ERR");
    fnmerge(currentBIN, fn.drive, fn.dir, fn.name, ".BIN");
    fnmerge(currentFN,  fn.drive, fn.dir, fn.name, "");

    if(errWindow) {
        errWindow->close();
        }

    // Assembler aufrufen, es sei denn, es existiert bereits ein
    // aktueller Assembler-Output; unbedingt aufrufen bei cmRebuild
    if( !fileexist(currentBIN) || (ftcmp(currentSRC,currentBIN) > 0) ||
    (cmd==cmRebuild) ) {
        suspend();
        sprintf(doscmd,"A! %s %04x",currentFN,PrgStart);
        system(doscmd);
        resume();
        redraw();
```

```
        if(fileexist(currentERR)) { /* errors occured */
            errWindow = (UErrViewer *) validView(new
UErrViewer(rErrWin,currentERR));
            deskTop->insert(errWindow);
            return("Assembler error(s)!");
            }

        if((resultInfo=sr2bin(currentOBJ, currentBIN, PrgStart, PrgEnd))!=NULL)
            if(*resultInfo == '\t') {
                resultInfo++;
                execDialog(createInfoDlg(resultInfo),0);
                }
            else
                return(resultInfo);
        }
    else
        if(cmd == cmAssemble) return("Bin file is up to date.");

    if(cmd == cmAssemble || cmd == cmRebuild) return("Assembler: Success.");


    // wenn Simulation gewählt, dann externen "TESTE68" aufrufen
    if(cmd == cmSimulate) {
        suspend();
        sprintf(doscmd,"S! %s",currentFN);
        system(doscmd);
        resume();
        redraw();
        return("Simulator closed.");
        }

    // EEPROM programmieren und starten, wenn EEPROM der Zielspeicher ist
    if(TargetOptions.Loadprg) {
        if(StatusWindow) {
            StatusWindow->close();
            }
        StatusWindow = createInfoWin("Programming EEPROM, please wait...");
        deskTop->insert(StatusWindow);
        if(ter->progEEPROM(currentBIN, PrgSize, PrgStart) == False)
            return(ter->getErrorDesc());
        if(ter->startEEPROM() == False)
            return(ter->getErrorDesc());
        }

    // Programm ins RAM laden und starten, wenn RAM der Zielspeicher ist
    else {
        if(StatusWindow) {
            StatusWindow->close();
            }
        StatusWindow = createInfoWin("Loading RAM, please wait...");
        deskTop->insert(StatusWindow);
        if( ter->loadANDstartRAM(currentBIN) == False)
            return(ter->getErrorDesc());
        }
    return("Target: Executing...");
    }
```

```
/*-- HC11IDEapp::trace() ------------------------------------------------*/

void HC11IDEapp::trace() {

    if(ter->loadANDstartRAM("TRACE.BIN") == False) {
        execDialog(createInfoDlg("Target hardware is not responding!"), 0);
        return;
        }
    while(execDialog(createTraceDlg(),0) != cmCancel) ;
    }


/*--HC11IDEapp::targetOpts() -----------------------------------------------*/

void HC11IDEapp::targetOpts() {

    if(execDialog(createTargOptDlg(),&TargetOptions) != cmCancel) {
        PrgStart = 0x0000;          /* RAM! */
        PrgEnd   = 0x00ff;
        PrgSize  = 0x0100;
        if(TargetOptions.Loadprg) {     /* EEPROM! */
            PrgStart = 0xb600;
            PrgEnd   = 0xb7ff;
            PrgSize  = 0x0200;
            if(TargetOptions.MCUtype) { /* E2! */
                PrgStart = 0xf800;
                PrgEnd   = 0xffff;
                PrgSize  = 0x0800;
                }
            }
        message(this,evCommand,cmOptionsModified,NULL);
        }
    }


/*-- HC11IDEapp::about() ------------------------------------------------*/

void HC11IDEapp::about() {

    execDialog(createAboutDlg(),0);
    }


/*-- HC11IDEapp::handleEvent() ------------------------------------------*/

void HC11IDEapp::handleEvent( TEvent& event )
{

    TApplication::handleEvent( event );
    if( event.what != evCommand )
        return;
    else
        switch(event.message.command)
            {
            case cmOpen:
                fileOpen();
                break;
```

```
case cmNew:
    fileNew();
    break;

case cmChangeDrct:
    changeDir();
    break;

case cmDosShell:
    dosShell();
    break;

case cmShowClip:
    showClip();
    break;

case cmTile:
    tile();
    break;

case cmCascade:
    cascade();
    break;

case cmAssemble:
case cmRebuild:
case cmExecute:
case cmSimulate:
    const char *result = make(event.message.command);
    if(StatusWindow) {
        StatusWindow->close();
        StatusWindow = NULL;
        }
    execDialog(createInfoDlg(result),0);
    break;

case cmTrace:
    trace();
    break;

case cmTraceFrom:
    char inp[20];
    unsigned cmd_state, trace_adr, save_PrgStart;

    sprintf(inp,"%04x",PrgStart);
    do {
        cmd_state = execDialog(createTraceFromDlg(), inp);
        if(cmd_state == cmCancel) {
            clearEvent(event);
            return;
            }
        } while(sscanf(inp,"%x",&trace_adr) != 1);
    save_PrgStart = PrgStart;
    PrgStart = trace_adr;
    trace();
    PrgStart = save_PrgStart;
    break;
```

```
            case cmTargetopts:
                targetOpts();
                break;

            case cmAboutCmd:
                about();
                break;

            default:
                return;
            }
    clearEvent( event );
    }


/*--- analyze_commandline() ------------------------------------------------*/

void analyze_commandline(int argc,char **argv) {
    int i;

    menuState = 0;

    for (i=0; i<argc; i++) {
        if(!i) continue;
        if( (argv[i][0] == '-') || (argv[i][0] == '/') ) {
            switch(toupper(argv[i][1])) {
                case 'S':
                    menuState = 1;
                    break;
                }
            }
        }
    }


/*--- main() ---------------------------------------------------------------*/

int main(int argc, char **argv) {
    analyze_commandline(argc,argv);
    HC11IDEapp ideApp;
    ideApp.run();
    return 0;
    }


/*--- fine -----------------------------------------------------------------*/
```

D.3. HC_DLG.CPP

```
/*=========================================================================*/
/*                                                                         */
/*  Titel:    HC_DLG.CPP                                                   */
/*  Funktion: Dialog-Generierung für die IDE                               */
/*  Version:  1.00                                                         */
/*  Datum:    22.2.94                                                      */
/*  Autor:    Oliver Thamm                                                 */
/*                                                                         */
/*=========================================================================*/


/*-- defines ------------------------------------------------------------*/


/*-- includes -----------------------------------------------------------*/

#include "hc11ide.h"


/*-- execDialog() -------------------------------------------------------*/
//
//  executes a dialog, destroys TDialog object after execution
//  and sets / gets data, if used
//
ushort execDialog( TDialog *d, void *data ) {

    TView *p = TProgram::application->validView( d );
    if( p == 0 )
        return cmCancel;
    else
        {
        if( data != 0 )
            p->setData( data );
        ushort result = TProgram::deskTop->execView( p );
        if( result != cmCancel && data != 0 )
            p->getData( data );
        TObject::destroy( p );
        return result;
        }
    }


/*-- createFindDialog() -------------------------------------------------*/

TDialog *createFindDialog() {

    TDialog *d = new TDialog( TRect( 0, 0, 38, 12 ), "Find" );

    d->options |= ofCentered;

    TInputLine *control = new TInputLine( TRect( 3, 3, 32, 4 ), 80 );
    d->insert( control );
    d->insert(
        new TLabel( TRect( 2, 2, 15, 3 ), "~T~ext to find", control ) );
    d->insert(
```

```
        new THistory( TRect( 32, 3, 35, 4 ), control, 10 ) );

    d->insert( new TCheckBoxes( TRect( 3, 5, 35, 7 ),
        new TSItem( "~C~ase sensitive",
        new TSItem( "~W~hole words only", 0 ))));

    d->insert(
        new TButton( TRect( 14, 9, 24, 11 ), "O~K~", cmOK, bfDefault ) );
    d->insert(
        new TButton( TRect( 26, 9, 36, 11 ), "Cancel", cmCancel, bfNormal));

    d->selectNext( False );
    return d;
    }


/*-- createReplaceDialog() ------------------------------------------------*/

TDialog *createReplaceDialog() {

    TDialog *d = new TDialog( TRect( 0, 0, 40, 16 ), "Replace" );

    d->options |= ofCentered;

    TInputLine *control = new TInputLine( TRect( 3, 3, 34, 4 ), 80 );
    d->insert( control );
    d->insert(
        new TLabel( TRect( 2, 2, 15, 3 ), "~T~ext to find", control ) );
    d->insert( new THistory( TRect( 34, 3, 37, 4 ), control, 10 ) );

    control = new TInputLine( TRect( 3, 6, 34, 7 ), 80 );
    d->insert( control );
    d->insert( new TLabel( TRect( 2, 5, 12, 6 ), "~N~ew text", control ) );
    d->insert( new THistory( TRect( 34, 6, 37, 7 ), control, 11 ) );

    d->insert( new TCheckBoxes( TRect( 3, 8, 37, 12 ),
        new TSItem("~C~ase sensitive",
        new TSItem("~W~hole words only",
        new TSItem("~P~rompt on replace",
        new TSItem("~R~eplace all", 0 ))))));

    d->insert(
        new TButton( TRect( 17, 13, 27, 15 ), "O~K~", cmOK, bfDefault ) );
    d->insert( new TButton( TRect( 28, 13, 38, 15 ),
                            "Cancel", cmCancel, bfNormal ) );

    d->selectNext( False );

    return d;
    }


/*-- createInfoWin() ------------------------------------------------------*/

TWindow *createInfoWin(const char *msg) {

    TWindow *w = new TWindow(TRect(0,0,strlen(msg)+8,7),"Info",0);
```

```
    w->options |= ofCentered;
    w->flags &= ~(wfClose | wfZoom);
    TView *tv1 = new TStaticText( TRect(0,0,strlen(msg),1), msg);
    tv1->options |= ofCentered;
    w->insert(tv1);
    return(w);
    }


/*-- createInfoDlg() -------------------------------------------------------*/

TDialog *createInfoDlg(const char *msg) {

    TDialog *d = new TDialog(TRect(0,0,strlen(msg)+6,7),"Info");

    d->options |= ofCentered;
    TView *tv1 = new TStaticText( TRect(0,2,strlen(msg),3), msg);
    tv1->options |= ofCenterX;
    d->insert(tv1);
    TView *tv2 = new TButton(TRect(0,4,10,6), "~O~K", cmCancel, bfDefault);
    tv2->options |= ofCenterX;
    d->insert(tv2);
    return(d);
    }


/*-- createTraceDlg() ------------------------------------------------------*/

TDialog *createTraceDlg() {

    TDialog *d = new TDialog(TRect(0,0,35,17),"Trace");

    d->options |= ofCentered;
    d->insert(
        new UTraceView( TRect(1,1,34,14) ));
    d->insert(
        new TButton( TRect(23,14,33,16),"~C~ancel", cmCancel, bfNormal ) );
    d->insert(
        new TButton( TRect(11,14,22,16),"~S~tep" , cmOK, bfDefault ) );
    return(d);
    }


/*--createAboutDlg() ----------------------------------------------------*/

TDialog *createAboutDlg() {

    TDialog *aboutBox = new TDialog(TRect(0, 0, 60, 17), "About");
    aboutBox->insert(new TStaticText(TRect(1, 2, 58, 9),
        "\003HC11 INTEGRATED DEVELOPMENT ENVIRONMENT\n"
        "\003DiplomWare Version 1.00 02/22/94\n"
        "\003\n"
        "\003Written by Oliver Thamm\n"
        "\003Copyright (C)1993,94 by HTWK Leipzig (FH)\n"
        "\003FG Mikrorechentechnik\n"
        "\003\n"
        ) );
    aboutBox->insert(new TStaticText(TRect(1, 9, 58, 13),
```

```
        "\003Created using Turbo Vision (C++ Version)\n"
        "\003by Borland International\n"
        "\003\n"
        "\003This version of HC11IDE is free, enjoy it!\n"
        ) );
    aboutBox->insert(new TButton(TRect(25, 14, 35, 16),"OK",cmOK,bfDefault));
    aboutBox->options |= ofCentered;
    return(aboutBox);
    }


/*-- createTargetOptDlg() ------------------------------------------------*/

TDialog *createTargOptDlg() {

    TDialog *dlg = new TDialog(TRect(0,0,33,10),"Target Options");
    dlg->options |= ofCentered;

    TView *tv1 = new TRadioButtons( TRect(3,3,16,5),
                    new TSItem("HC11~A~1",
                    new TSItem("HC811~E~2",0)));
    dlg->insert(tv1);
    dlg->insert(new TLabel(TRect(2,2,11,3),"MCU type",tv1));

    TView *tv2 = new TRadioButtons( TRect(18,3,30,5),
                    new TSItem("~R~AM",
                    new TSItem("EE~P~ROM",0)));
    dlg->insert(tv2);
    dlg->insert(new TLabel(TRect(17,2,26,3),"Load prg",tv2));

    dlg->insert(new TButton(TRect(20,7,30,9),"~C~ancel",cmCancel,bfNormal));
    dlg->insert(new TButton(TRect(8,7,18,9),"~O~K",cmOK,bfDefault));

    return(dlg);
    }


/*-- createTraceFromDlg() ------------------------------------------------*/

TDialog *createTraceFromDlg() {

    TDialog *d = new TDialog( TRect( 0, 0, 29, 7 ), "Trace from" );

    d->options |= ofCentered;

    TInputLine *control = new TInputLine( TRect( 19, 2, 26, 3 ), 5 );
    d->insert( control );
    d->insert(
        new TLabel( TRect( 2, 2, 18, 3 ), "~S~tart at (hex):", control ) );
    d->insert(
        new TButton( TRect( 5, 4, 15, 6 ), "O~K~", cmOK, bfDefault ) );
    d->insert(
        new TButton( TRect( 17, 4, 27, 6 ), "Cancel", cmCancel, bfNormal));

    d->selectNext( False ); // selects OK-Button
    return d;
    }
```

```
/*-- doEditDialog() ----------------------------------------------------------*/

typedef char *_charPtr;
typedef TPoint *PPoint;

#pragma warn -rvl

ushort doEditDialog( int dialog, ... )
{
    va_list arg;

    char buf[80];
    ostrstream os( buf, sizeof( buf ) );
    switch( dialog )
        {
        case edOutOfMemory:
            return messageBox( "Not enough memory for this operation",
                               mfError | mfOKButton );
        case edReadError:
            {
            va_start( arg, dialog );
            os << "Error reading file " << va_arg( arg, _charPtr )
               << "." << ends;
            va_end( arg );
            return messageBox( buf, mfError | mfOKButton );
            }
        case edWriteError:
            {
            va_start( arg, dialog );
            os << "Error writing file " << va_arg( arg,_charPtr )
               << "." << ends;
            va_end( arg );
            return messageBox( buf, mfError | mfOKButton );
            }
        case edCreateError:
            {
            va_start( arg, dialog );
            os << "Error creating file " << va_arg( arg, _charPtr )
               << "." << ends;
            va_end( arg );
            return messageBox( buf, mfError | mfOKButton );
            }
        case edSaveModify:
            {
            va_start( arg, dialog );
            os << va_arg( arg, _charPtr )
               << " has been modified. Save?" << ends;
            va_end( arg );
            return messageBox( buf, mfInformation | mfYesNoCancel );
            }
        case edSaveUntitled:
            return messageBox( "Save untitled file?",
                               mfInformation | mfYesNoCancel );
        case edSaveAs:
            {
            va_start( arg, dialog );
            return execDialog( new TFileDialog( "*.*",
```

```
                                                "Save file as",
                                                "~N~ame",
                                                fdOKButton,
                                                101 ),va_arg(arg,_charPtr));
            }

        case edFind:
            {
            va_start( arg, dialog );
            return execDialog( createFindDialog(), va_arg( arg, _charPtr ) );
            }

        case edSearchFailed:
            return messageBox( "Search string not found.",
                               mfError | mfOKButton );
        case edReplace:
            {
            va_start( arg, dialog );
            return execDialog( createReplaceDialog(), va_arg(arg,_charPtr));
            }

        case edReplacePrompt:
            //  Avoid placing the dialog on the same line as the cursor
            TRect r( 0, 1, 40, 8 );
            r.move( (TProgram::deskTop->size.x-r.b.x)/2, 0 );
            TPoint t = TProgram::deskTop->makeGlobal( r.b );
            t.y++;
            va_start( arg, dialog );
            TPoint *pt = va_arg( arg, PPoint );
            if( pt->y <= t.y )
                r.move( 0, TProgram::deskTop->size.y - r.b.y - 2 );
            va_end( arg );
            return messageBoxRect( r, "Replace this occurence?",
                                   mfYesNoCancel | mfInformation );

        }
}

#pragma warn .rvl


/*-- fine ----------------------------------------------------------------*/
```

D.4. HC_MENU.CPP

```
/*=========================================================================*/
/*                                                                         */
/*  Titel:    HC_MENU.CPP                                                  */
/*  Funktion: enthält Menü und Statuszeilen Definitionen sowie einige      */
/*            (globale) Hilfsfunktionen                                    */
/*  Version:  1.00                                                         */
/*  Datum:    22.2.94                                                      */
/*  Autor:    Oliver Thamm                                                 */
/*                                                                         */
/*=========================================================================*/


/*-- defines ------------------------------------------------------------*/


/*-- includes -----------------------------------------------------------*/

#include "hc11ide.h"


/*-- externals ----------------------------------------------------------*/

extern unsigned menuState;


/*-- HC11IDEapp::initMenuBar() ------------------------------------------*/

TMenuBar *HC11IDEapp::initMenuBar( TRect r ) {

  TSubMenu& sub0 = *new TSubMenu( "~\xf0~", kbAltSpace ) +
    *new TMenuItem( "~A~bout...", cmAboutCmd, kbNoKey );

  TSubMenu& sub1 = *new TSubMenu( "~F~ile", kbAltF ) +
    *new TMenuItem( "~O~pen...", cmOpen, kbF3, hcNoContext, "F3" ) +
    *new TMenuItem( "~N~ew", cmNew, kbNoKey ) +
    *new TMenuItem( "~S~ave", cmSave, kbF2, hcNoContext, "F2" ) +
    *new TMenuItem( "S~a~ve as...", cmSaveAs, kbNoKey ) +
    *new TMenuItem( "C~l~ose", cmClose, kbAltF3, hcNoContext, "Alt-F3" ) +
         newLine() +
    *new TMenuItem( "~C~hange dir...", cmChangeDrct, kbNoKey ) +
    *new TMenuItem( "~D~OS shell", cmDosShell, kbNoKey ) +
    *new TMenuItem( "E~x~it", cmQuit, kbAltX, hcNoContext, "Alt-X" );

  TSubMenu& sub2 = *new TSubMenu( "~E~dit", kbAltE ) +
    *new TMenuItem( "~U~ndo", cmUndo, kbNoKey ) +
         newLine() +
    *new TMenuItem( "Cu~t~", cmCut, kbShiftDel, hcNoContext, "Shift-Del" ) +
    *new TMenuItem( "~C~opy", cmCopy, kbCtrlIns, hcNoContext, "Ctrl-Ins" ) +
    *new TMenuItem( "~P~aste", cmPaste,kbShiftIns,hcNoContext,"Shift-Ins") +
    *new TMenuItem( "~S~how clipboard", cmShowClip, kbNoKey ) +
         newLine() +
    *new TMenuItem( "~C~lear", cmClear, kbCtrlDel, hcNoContext, "Ctrl-Del" );

  TSubMenu& sub3 = *new TSubMenu( "~S~earch", kbAltS ) +
    *new TMenuItem( "~F~ind...", cmFind, kbNoKey ) +
```

```
    *new TMenuItem( "~R~eplace...", cmReplace, kbNoKey ) +
    *new TMenuItem( "~S~earch again", cmSearchAgain, kbNoKey );

  TSubMenu& sub4 = *new TSubMenu( "~T~arget", kbAltT ) +
    *new TMenuItem( "~A~ssemble  F9", cmAssemble, kbF9 ) +
    *new TMenuItem( "~R~ebuild     ", cmRebuild,  kbNoKey ) +
    *new TMenuItem( "~E~xecute   F8", cmExecute,  kbF8 );

  if(menuState == 1) {
    sub4 = sub4 + *new TMenuItem( "~S~imulate    ", cmSimulate, kbNoKey );
    }

  TSubMenu& sub5 = *new TSubMenu( "~D~ebug", kbAltD ) +
    *new TMenuItem( "~T~race      F7", cmTrace, kbF7 ) +
    *new TMenuItem( "Trace ~f~rom...", cmTraceFrom, kbNoKey );

  TSubMenu& sub6 = *new TSubMenu( "~O~ptions", kbAltO ) +
    *new TMenuItem( "~H~ardware...", cmTargetopts, kbNoKey );

  r.b.y = r.a.y+1;
  return new TMenuBar( r, sub0 + sub1 + sub2 + sub3 + sub4 + sub5 + sub6 );
  }


/*-- HC11IDEapp::initStatusLine() ------------------------------------------*/

TStatusLine *HC11IDEapp::initStatusLine( TRect r ) {

    r.a.y = r.b.y-1;
    return new TStatusLine( r,
        *new TStatusDef( 0, 0xFFFF ) +
            *new TStatusItem("~F2~ Save",      kbF2,    cmSave    ) +
            *new TStatusItem("~F3~ Open",      kbF3,    cmOpen    ) +
            *new TStatusItem("~F9~ Assemble",  kbF9,    cmAssemble) +
            *new TStatusItem("~F8~ Execute",   kbF8,    cmExecute ) +
            *new TStatusItem("~F7~ Trace",     kbF7,    cmTrace   ) +
            *new TStatusItem("~ALT-X~ Exit",   kbAltX,  cmQuit    )
            );
    }


/*-- HC11IDEapp::outOfMemory() ---------------------------------------------*/

void HC11IDEapp::outOfMemory() {

    messageBox("Not enough memory for this operation.",mfError|mfOKButton );
    }


/*-- ftcmp() ---------------------------------------------------------------*/

int ftcmp(const char *fn1, const char *fn2) {

    ffblk ff1, ff2;

    if(findfirst(fn1,&ff1,0)) return(0); /* file not found */
    if(findfirst(fn2,&ff2,0)) return(0); /* file not found */
    if(ff1.ff_fdate == ff2.ff_fdate)
```

```
        return(ff1.ff_ftime - ff2.ff_ftime);
    return(ff1.ff_fdate - ff2.ff_fdate);
    }


/*-- fileexist() -------------------------------------------------------------*/

Boolean fileexist(const char *fn) {

    ffblk fff;

    return((Boolean)!findfirst(fn,&fff,0));
    }


/*-- path_open() -------------------------------------------------------------*/

FILE * path_fopen(const char *fn, const char *mode) {

    FILE *fp;
    char *fullName, nameBuffer[120], *envPath;

    if((fp = fopen(fn, mode)) != NULL)  // full path given or file in
        return(fp);                     // current directory

    envPath = getenv("PATH");
    if(!envPath) return(NULL);

    while(1) {
        fullName = nameBuffer;
        while((*envPath != ';') && (*envPath != '\0'))
            *fullName++ = *envPath++;
        *fullName = '\0';
        strcat(nameBuffer,"\\");
        strcat(nameBuffer,fn);

        if((fp = fopen(nameBuffer,mode)) != NULL)
            return(fp);

        if(*envPath) envPath++;
        else return(NULL);
        }
    }


/*-- fine --------------------------------------------------------------------*/
```

D.5. HC_TERM.CPP

```
/*=============================================================================*/
/*                                                                             */
/*  Titel:    HC_TERM.CPP                                                      */
/*  Funktion: UTerminal member functions für serielle Kommunikation            */
/*  Version:  1.00                                                             */
/*  Datum:    22.2.94                                                          */
/*  Autor:    Oliver Thamm                                                     */
/*                                                                             */
/*=============================================================================*/


/*-- defines ----------------------------------------------------------------*/


/*-- includes ---------------------------------------------------------------*/

#include "hc11ide.h"


/*-- externals --------------------------------------------------------------*/

extern unsigned PrgStart;



/*-- UTerminal constructor --------------------------------------------------*/
//
//  ComPort = 1...4
//
UTerminal::UTerminal(unsigned BaudRate, unsigned ComPort) {

    reinitialize(BaudRate,ComPort);
    }


/*-- UTerminal::reinitialize() ----------------------------------------------*/

void UTerminal::reinitialize(unsigned BaudRate, unsigned ComPort) {

    unsigned long b;

    theComPort = (ComPort-1) & 0x03;
    thePortBase = *((unsigned far *) MK_FP(0x0040, theComPort<<1 ));
    bioscom(_COM_INIT, _COM_9600 | _COM_CHR8 | _COM_STOP1 | _COM_NOPARITY,
theComPort);
    b = ((1152000l / BaudRate) + 5) / 10;
    outportb(thePortBase+3, inportb(thePortBase+3) | 0x80);  /* set DLAB */
    outportb(thePortBase+0, b & 0xff);
    outportb(thePortBase+1, b >> 8  );
    outportb(thePortBase+3, inportb(thePortBase+3) & ~0x80); /* res DLAB */
    }


/*-- UTerminal::getErrorDesc() ----------------------------------------------*/
//
```

```
//  returns a pointer to the description string
//
const char * UTerminal::getErrorDesc() {

    static const char *s;

    switch(errordesc) {
        case edResetFailed:
            s = "Reset failed!";
            break;
        case edTxTimeout:
            s = "Transmitter timeout!";
            break;
        case edDlFileNotFnd:
            s = "Download file not found!";
            break;
        case edEOF:
            s = "Unexpected EOF!";
            break;
        case edHndshakeFailed:
            s = "Download handshake failed!";
            break;
        case edRamFile2Long:
            s = "Download file > 256 bytes!";
            break;
        case edPgmError:
            s = "Programming error!";
            break;
        case edEepFile2Long:
            s = "Binary file > EEPROM size!";
            break;
        default:
            s = "Error!";
            break;
        }
    return(s);
    }


/*-- UTerminal::getCOM() ---------------------------------------------------*/
//
//  get one char from COM port
//  returns -1 if no char available
//
int UTerminal::getCOM() {

    int n;

    for(n=0; n<20; n++) {
        if(bioscom(_COM_STATUS, 0, theComPort) & _COM_RxDataAvail)
            return(inportb(thePortBase));
        delay(1);
        }
    return(-1);
    }


/*-- UTerminal::putCOM() ---------------------------------------------------*/
```

```
//
//  send one char to COM port
//  if TX buffer isn't empty, try another 10 times
//
Boolean UTerminal::putCOM(char c) {

    int n;

    for(n=0; n<20; n++) {
        if(bioscom(_COM_STATUS, 0, theComPort) & _COM_TxBufEmpty) {
            outportb(thePortBase,c);
            return(True);
            }
        delay(1);
        }
    return(False);
    }


/*-- UTerminal::resetTarget() ------------------------------------------------*/
//
//  pull RTS Low for 2 ms
//
Boolean UTerminal::resetTarget() {

    outportb(thePortBase+4, inportb(thePortBase+4) | 0x01 );
    delay(2);
    outportb(thePortBase+4, inportb(thePortBase+4) & 0xfe );
    delay(20);
    if(testBreak()) {
        getCOM();
        return(True);
        }
    return(False);
    }


/*-- UTerminal::testBreak() --------------------------------------------------*/
//
//  check for BREAK-detection
//
Boolean UTerminal::testBreak() {

    if(bioscom(_COM_STATUS, 0, theComPort) & _COM_BrkDetect)
        return(True);
    return(False);
    }


/*-- UTerminal::boot() -------------------------------------------------------*/
//
//  reset HC11, send a char (0x00/0x55/0xff)
//
Boolean UTerminal::boot(unsigned char c) {

    if(resetTarget() == False) {
        errordesc = edResetFailed;
        return(False);
```

```
        }
    if(putCOM(c) == False) {
        errordesc = edTxTimeout;
        return(False);
        }
    return(True);
    }


/*-- UTerminal::startRAM() -------------------------------------------------*/
//
//  reset HC11, send 0x55 (skip download, start RAM)
//
Boolean UTerminal::startRAM() {

    return(boot(0x55));
    }


/*-- UTerminal::startEEPROM() ----------------------------------------------*/
//
//  reset HC11, send 0x00 (skip download, start EEPROM)
//
Boolean UTerminal::startEEPROM() {

    return(boot(0x00));
    }


/*-- UTerminal::loadANDstartRAM() ------------------------------------------*/
//
//  reset HC11, send 0xff, download 256 byte user prg into RAM
//  (and start RAM)
//
Boolean UTerminal::loadANDstartRAM(const char *filename) {

    int c, n;
    FILE *fp;


    if(boot(0xff) == False)
        return(False);

    if((fp=path_fopen(filename,"rb")) == NULL) {
        errordesc = edDlFileNotFnd;
        return(False);
        }
    for(n=0; n<256; n++) {
        if((c=getc(fp)) == EOF) {
            c = 0;
            }
        if(putCOM(c) == False) {
            errordesc = edTxTimeout;
            fclose(fp);
            return(False);
            }
        if(getCOM() != c) {
            errordesc = edHndshakeFailed;
```

```
            fclose(fp);
            return(False);
            }
        }
    if(getc(fp) != EOF) {
        errordesc = edRamFile2Long;
        fclose(fp);
        return(False);
        }
    fclose(fp);
    return(True);
    }


/*-- UTerminal::progEEPROM() ------------------------------------------------*/
//
//  reset HC11, send 0xff, download EEPROMer prg into RAM,
//  start EEPROMer prg (which awaits n bytes user prg to burn into EEPROM)
//
Boolean UTerminal::progEEPROM(const char *filename,
    unsigned eeprom_size, unsigned eeprom_start) {

    int c, n;
    FILE *fp;

    if(loadANDstartRAM("EEPROG.BIN") == False)
        return(False);
    delay(20);
    putCOM(eeprom_start & 0x00ff);
    putCOM(eeprom_start >> 8);

    if((fp=fopen(filename,"rb")) == NULL) {
        errordesc = edDlFileNotFnd;
        return(False);
        }
    for(n=0; n < eeprom_size; n++) {
        if((c=getc(fp)) == EOF) {
            break;
            }
        if(putCOM(c) == False) {
            errordesc = edTxTimeout;
            fclose(fp);
            return(False);
            }
        delay(15);
        if(getCOM() != c) {
            errordesc = edPgmError;
            fclose(fp);
            return(False);
            }
        }
    if(getc(fp) != EOF) {
        errordesc = edEepFile2Long;
        fclose(fp);
        return(False);
        }
    fclose(fp);
    return(True);
```

```
    }


/*-- UTerminal::step4step() ------------------------------------------------*/
//
//  trace mode: execute one prg step and get CPU-state
//
Boolean UTerminal::step4step(UTraceInfo *TraceInfo) {

    putCOM(PrgStart & 0x00ff);
    putCOM(PrgStart >> 8);

    TraceInfo->CCR   = getCOM();
    TraceInfo->CODE[0] = getCOM();
    TraceInfo->ACCB  = getCOM();
    TraceInfo->CODE[1] = getCOM();
    TraceInfo->ACCA  = getCOM();
    TraceInfo->CODE[2] = getCOM();
    TraceInfo->IXH   = getCOM();
    TraceInfo->CODE[3] = getCOM();
    TraceInfo->IXL   = getCOM();
    TraceInfo->CODE[4] = getCOM();
    TraceInfo->IYH   = getCOM();
    TraceInfo->CODE[5] = getCOM();
    TraceInfo->IYL   = getCOM();
    TraceInfo->CODE[6] = getCOM();
    TraceInfo->PCH   = getCOM();
    TraceInfo->CODE[7] = getCOM();
    TraceInfo->PCL   = getCOM();
    TraceInfo->CODE[8] = getCOM();
    TraceInfo->SPH   = getCOM();
    TraceInfo->SPL   = getCOM();
    TraceInfo->PORTA = getCOM();
    TraceInfo->PORTB = getCOM();
    TraceInfo->PORTC = getCOM();
    TraceInfo->PORTD = getCOM();
    TraceInfo->PORTE = getCOM();

    if(getCOM() != '\n')
            return(False);
    return(True);
    }


/*-- fine ------------------------------------------------------------------*/
```

D.6. HC_REAS.CPP

```
/*============================================================================*/
/*                                                                            */
/*  Titel:    HC_REAS.CPP                                                     */
/*  Funktion: 68HC11 Reassembler                                              */
/*  Version:  1.00                                                            */
/*  Datum:    22.2.94                                                         */
/*  Autor:    Oliver Thamm                                                    */
/*                                                                            */
/*============================================================================*/


/*-- includes ----------------------------------------------------------------*/

#include "hc11ide.h"


/*-- globals -----------------------------------------------------------------*/

static char * mnemo_0[4][16] = {
    {   "TEST", "NOP",  "IDIV", "FDIV", "LSRD", "ASLD", "TAP",  "TPA",
        "INX",  "DEX",  "CLV",  "SEV",  "CLC",  "SEC",  "CLI",  "SEI"   },
    {   "SBA",  "CBA",  "BRSET","BRCLR","BSET", "BCLR", "TAB",  "TBA",
        NULL,   "DAA",  NULL,   "ABA",  "BSET", "BCLR", "BRSET","BRCLR" },
    {   "BRA",  "BRN",  "BHI",  "BLS",  "BCC",  "BCS",  "BNE",  "BEQ",
        "BVC",  "BVS",  "BPL",  "BMI",  "BGE",  "BLT",  "BGT",  "BLE"   },
    {   "TSX",  "INS",  "PULA", "PULB", "DES",  "TXS",  "PSHA", "PSHB",
        "PULX", "RTS",  "ABX",  "RTI",  "PSHX", "MUL",  "WAI",  "SWI"   } };

static char * mnemo_4[16] = {
        "NEG",  NULL,   NULL,   "COM",  "LSR",  NULL,   "ROR",  "ASR",
        "ASL",  "ROL",  "DEC",  NULL,   "INC",  "TST",  "JMP",  "CLR"   };

static char * mnemo_8[16] = {
        "SUBA", "CMPA", "SBCA", "SUBD", "ANDA", "BITA", "LDAA", "STAA",
        "EORA", "ADCA", "ORAA", "ADDA", "CPX",  "JSR",  "LDS",  "STS"   };

static char * mnemo_c[16] = {
        "SUBB", "CMPB", "SBCB", "ADDD", "ANDB", "BITB", "LDAB", "STAB",
        "EORB", "ADCB", "ORAB", "ADDB", "LDD",  "STD",  "LDX",  "STX"   };


/*-- UReAssembler::x2y() -----------------------------------------------------*/
//
// change X-related mnemonic to Y
// return zero if error
//
int UReAssembler::x2y(char *s) {

    int replaced = 0;
    char *p = s + strlen(s);

    while(p > (s+1)) {  // leave out 1st character because XGDX
        p--;
        if(*p == 'X') {
            *p = 'Y';
```

```
            replaced = 1;
            }
        }
    return(replaced);
    }


/*-- UReAssembler::getMnemonic() ---------------------------------------*/
//
// returns mnemonic string
// or NULL, if illegal opcode
//
char * UReAssembler::getMnemonic(unsigned char opcode, unsigned char& prefix) {

    static char mBuf[10];

    if(prefix == 0x1a) {
        switch(opcode) {
            case 0x83:      // Exception CPD statt SUBD,
            case 0x93:      // keine weiteren Effekte des Präfix
            case 0xa3:
            case 0xb3:
                prefix = 0;
                return("CPD");
            case 0xac:
                prefix = 0;
                return("CPY");
            case 0xee:
                prefix = 0;
                return("LDY");
            case 0xef:
                prefix = 0;
                return("STY");
            default:
                return(NULL);
            }
        }
    if(prefix == 0xcd) {
        switch(opcode) {    // Exception CPD statt SUBD,
            case 0xa3:      // Präfix mit unveränderter Wirkung auf Operand
                return("CPD");
            case 0xac:
            case 0xee:
            case 0xef:
                break;
            default:
                return(NULL);
            }
        }
    switch(opcode & 0xc0) {
        case 0x00:
            return(mnemo_0[opcode >> 4][opcode & 0x0f]);
        case 0x40:
            if((opcode == 0x4e) || (opcode == 0x5e)) return(NULL);
            strcpy(mBuf,mnemo_4[opcode & 0x0f]);
            if((opcode & 0xf0) == 0x40) strcat(mBuf,"A");
            if((opcode & 0xf0) == 0x50) strcat(mBuf,"B");
            return(mBuf);
```

```
        case 0x80:
            if(opcode == 0x87) return(NULL);
            if(opcode == 0x8d) return("BSR");
            if(opcode == 0x8f) return("XGDX");
            return(mnemo_8[opcode & 0x0f]);
        case 0xc0:
            if((opcode == 0xc7) || (opcode == 0xcd)) return(NULL);
            if(opcode == 0xcf) return("STOP");
            return(mnemo_c[opcode & 0x0f]);
        }
    return(NULL);
    }


/*-- UReAssembler::opRel() ------------------------------------------------*/
//
// relative addressing mode
//
char * UReAssembler::opRel(unsigned pc, unsigned offset) {

    static char b[6];
    unsigned destination;

    if(offset & 0xff00) return(NULL);
    destination = pc + offset;
    if(offset > 127) destination -= 256;
    sprintf(b,"$%04x",destination);
    return(b);
    }


/*-- UReAssembler::opImm() ------------------------------------------------*/
//
// immediate addressing mode
//
char * UReAssembler::opImm(unsigned value) {

    static char b[5];

    if(value & 0xff00) return(NULL);
    sprintf(b,"#$%02x",value);
    return(b);
    }


/*-- UReAssembler::opImm16() ----------------------------------------------*/
//
// immediate 16 bit addressing mode
//
char * UReAssembler::opImm16(unsigned value) {

    static char b[7];

    sprintf(b,"#$%04x",value);
    return(b);
    }
```

```
/*-- UReAssembler::opDir() ---------------------------------------------------*/
//
// direct addressing mode
//
char * UReAssembler::opDir(unsigned address) {

    static char b[7];

    if(address & 0xff00) return(NULL);
    sprintf(b,"<$00%02x",address);
    return(b);
    }


/*-- UReAssembler::opExt() ---------------------------------------------------*/
//
// extended addressing mode
//
char * UReAssembler::opExt(unsigned address) {

    static char b[6];

    sprintf(b,"$%04x",address);
    return(b);
    }


/*-- UReAssembler::opInd() ---------------------------------------------------*/
//
// indexed addressing mode
//
char * UReAssembler::opInd(unsigned index) {

    static char b[6];

    if(index & 0xff00) return(NULL);
    sprintf(b,"$%02x,X",index);
    return(b);
    }


/*-- UReAssembler::opMsk() ---------------------------------------------------*/
//
// mask operand
//
char * UReAssembler::opMsk(unsigned value) {

    static char b[11];
    int n;

    if(value & 0xff00) return(NULL);
    b[0] = '#';
    b[1] = '%';
    for(n=0; n<8; n++) {
        b[n+2] = (value & 0x80) ? '1' : '0';
        value <<= 1;
        }
    b[10] = '\0';
```

```
    return(b);
    }


/*-- UReAssembler::getOperands() ---------------------------------------*/
//
// returns operand string
// or NULL, if error
//
char * UReAssembler::getOperands(unsigned char opcode, unsigned long loperand,
unsigned pc) {

    static char b[20];
    unsigned operand;

#pragma warn -sig
    operand = loperand & 0xffff;
#pragma warn +sig

    switch(opcode >> 4) {
        case 0x00:
        case 0x03:
        case 0x04:
        case 0x05:
            return("");         // opInh()
        case 0x02:
            return(opRel(pc,operand));
        case 0x09:
        case 0x0d:
            return(opDir(operand));
        case 0x07:
        case 0x0b:
        case 0x0f:
            return(opExt(operand));
        case 0x06:
        case 0x0a:
        case 0x0e:
            return(opInd(operand));
        case 0x08:
        case 0x0c:
            if(opcode == 0x8d) return(opRel(pc,operand));
            if(opcode == 0x8f || opcode == 0xcf) return("");
            switch(opcode & 0x0f) {
                case 0x03:
                case 0x0c:
                case 0x0e:
                    return(opImm16(operand));
                }
            return(opImm(operand));
        case 0x01:
            switch(opcode & 0x0e) {
                case 0x00:  // INH
                case 0x06:
                case 0x08:
                case 0x0a:
                    *b = '\0';
                    break;
                case 0x02:  // DIR,MASK,REL
```

```
                    strcpy(b,opDir((unsigned)(loperand >> 16)));
                    strcat(b,",");
                    strcat(b,opMsk(operand >> 8));
                    strcat(b,",");
                    strcat(b,opRel(pc, operand & 0xff));
                    break;
                case 0x04:  // DIR,MASK
                    strcpy(b,opDir(operand >> 8));
                    strcat(b,",");
                    strcat(b,opMsk(operand & 0xff));
                    break;
                case 0x0c:  // IND,MASK
                    strcpy(b,opInd(operand >> 8));
                    strcat(b,",");
                    strcat(b,opMsk(operand & 0xff));
                    break;
                case 0x0e:  // IND,MASK,REL
                    strcpy(b,opInd((unsigned)(loperand >> 16)));
                    strcat(b,",");
                    strcat(b,opMsk(operand >> 8));
                    strcat(b,",");
                    strcat(b,opRel(pc, operand & 0xff));
                    break;
                }
            return(b);
        }
    return(NULL); // never happens
    }


/*-- UReAssembler::getByteCnt() ----------------------------------------------*/
//
// calculates number of (operand-) bytes for this opcode
//
unsigned UReAssembler::getByteCnt(unsigned char opcode) {

    switch(opcode >> 4) {
        case 0x00:
        case 0x03:
        case 0x04:
        case 0x05:
            return(0);
        case 0x02:
        case 0x06:
        case 0x09:
        case 0x0a:
        case 0x0d:
        case 0x0e:
            return(1);
        case 0x07:
        case 0x0b:
        case 0x0f:
            return(2);
        case 0x01:
            switch(opcode & 0x0e) {
                 case 0x00:
                 case 0x06:
                 case 0x08:
```

```
                case 0x0a:
                   return(0);
                case 0x04:
                case 0x0c:
                   return(2);
                case 0x02:
                case 0x0e:
                   return(3);
                }
            break;
        case 0x08:
        case 0x0c:
            if(opcode == 0x8f || opcode == 0xcf) return(0);
            switch(opcode & 0x0f) {
                case 0x03:
                case 0x0c:
                case 0x0e:
                    return(2);
                }
            return(1);
        }
    return(0); //never happens
    }


/*-- UReAssembler::reAssemble() ----------------------------------------------*/
//
// main function of the reassembler
// returns NULL in case of an error
// returns new pc, when successfull
//
unsigned UReAssembler::reAssemble(unsigned pc, unsigned char *data, char * buffer)
{

    char *mStr, *oStr;
    unsigned char prefix = 0;
    unsigned char opcode;
    unsigned new_pc = pc;
    unsigned byte_cnt;
    unsigned long operand = 0;

    opcode = *data++;
    if( (opcode == 0x18) || (opcode == 0x1a) || (opcode == 0xcd) ) {
        prefix = opcode;
        opcode = *data++;
        new_pc++;
        }

    mStr = getMnemonic(opcode, prefix);
    if(!mStr) return(NULL);
    strcpy(buffer, mStr);
    new_pc++;

    byte_cnt = getByteCnt(opcode);
    switch(byte_cnt) {
        case 3:
            operand = (unsigned long)(*data++) << 16;
            new_pc++;
```

```
        case 2:
            operand += (unsigned long)(*data++) << 8;
            new_pc++;
        case 1:
            operand += (unsigned long)(*data++);
            new_pc++;
            strcat(buffer," ");
        case 0:
            break;
        }

    oStr = getOperands(opcode, operand, new_pc);
    if(!oStr) return(0);
    if(prefix == 0xcd)
        if(!x2y(oStr)) return(NULL);
    strcat(buffer,oStr);
    if(prefix == 0x18)
        if(!x2y(buffer)) return(NULL);
    return(new_pc - pc);
    }


/*-- fine ----------------------------------------------------------------*/
```

D.7. HC_SR2B.CPP

```
/*============================================================================*/
/*                                                                            */
/*  Titel:    HC_SR2B.CPP                                                     */
/*  Funktion: Konvertiert eine Motorola S-Record-Datei in ein Binärfile       */
/*  Version:  1.00                                                            */
/*  Datum:    22.2.94                                                         */
/*  Autor:    Oliver Thamm                                                    */
/*                                                                            */
/*============================================================================*/


/*-- defines -----------------------------------------------------------------*/


/*-- includes ----------------------------------------------------------------*/

#include "hc11ide.h"


/*-- HC11IDEapp::hexbin() ----------------------------------------------------*/
//
//  converts a hexadecimal character to a binary number
//
unsigned HC11IDEapp::hexbin(char c) {

    if(c>='0' && c<='9')
        return(c-'0');
    if(c>='A' && c<='F')
        return(c-'A'+10);
    if(c>='a' && c<='f')
        return(c-'a'+10);
//    fprintf(stderr,"invalid hex digit\r\n");
    return(0);
    }


/*-- HC11IDEapp::hexe() ------------------------------------------------------*/
//
//  converts a hex-expression (string) to a 16 bit binary value
//
unsigned HC11IDEapp::hexe(char *s, unsigned sl) {

    unsigned i;
    unsigned long val = 0;

    for(i=0; i<sl; i++) {
        val += hexbin(*s++);
        val <<= 4;
        }
    return((unsigned)(val >> 4));
    }


/*-- HC11IDEapp::sr2bin() ----------------------------------------------------*/
//
```

```
//  reads out objfn and creates binfn
//  works only within the specified address range first ... last
//
const char * HC11IDEapp::sr2bin(const char *objfn, const char *binfn,
    unsigned first, unsigned last) {

    int n;
    unsigned RecLen;
    unsigned long currentAddr, recentAddr;
    FILE *inputfile;
    FILE *outputfile;
    char buffer[100];
    Boolean cut, empty;

    if((inputfile=fopen(objfn,"r"))==NULL) {
        return("Can't open obj file!");
        }
    if((outputfile=fopen(binfn,"wb"))==NULL) {
        return("Can't create bin file!");
        }

    cut = False;
    empty = True;
    recentAddr = 0l;
    while(1) {
        if(!fgets(buffer,99,inputfile)) break;
        if(buffer[0] != 'S') {
            return("Missing S-Record ID!");
            }
        if(buffer[1] == '9') break;
        if(buffer[1] == '0') continue;
        if(buffer[1] != '1') {
            return("Invalid S-Record type!");
            }
        RecLen = hexe(buffer+2,2)-3;
        currentAddr = hexe(buffer+4,4);
        if(currentAddr<recentAddr)          /* decending addresses ! */
            return("Overlapping code areas!");
        while(currentAddr>recentAddr) {     /* fill empty areas */
            if(recentAddr<first || recentAddr>last)
                ;
            else {
                putc(0xff,outputfile);
                }
            recentAddr++;
            }
        for(n=0; n<RecLen; n++, currentAddr++) {
            if(currentAddr<first || currentAddr>last)
                cut = True;
            else {
                putc(hexe(buffer+8+2*n,2),outputfile);
                empty = False;
                }
            }
        recentAddr = currentAddr;
        }

    fclose(inputfile);
```

```
    fclose(outputfile);
    if(empty)
        return("No code created!");
    if(cut)
        return("\tWarning: Output was cut!");
    return(NULL);
    }


/*-- fine ----------------------------------------------------------------*/
```

D.8. HC_HEAP.CPP

```
/*=========================================================================*/
/*                                                                         */
/*  Titel:    HC_HEAP.CPP                                                  */
/*  Funktion: THeapView member functions, zeigt den freien Speicherplatz an */
/*  Version:  1.00                                                         */
/*  Datum:    21.2.94                                                      */
/*  Autor:    Oliver Thamm                                                 */
/*                                                                         */
/*=========================================================================*/


/*-- defines ------------------------------------------------------------*/


/*-- includes -----------------------------------------------------------*/

#include "hc11ide.h"


/*-- THeapView constructor ----------------------------------------------*/

THeapView::THeapView(TRect& r) : TView( r ) {

    oldMem = 0;
    newMem = heapSize();
    }


/*-- THeapView::draw() --------------------------------------------------*/

void THeapView::draw() {

    TDrawBuffer buf;
    char c = getColor(2);

    buf.moveChar(0, ' ', c, size.x);
    buf.moveStr(0, heapStr, c);
    writeLine(0, 0, size.x, 1, buf);
    }


/*-- THeapView::update() ------------------------------------------------*/

void THeapView::update() {

    if( (newMem = heapSize()) != oldMem ) {
        oldMem = newMem;
        drawView();
        }
    }


/*-- THeapView::heapSize() ----------------------------------------------*/

long THeapView::heapSize() {
```

```
    long total = farcoreleft();
    struct farheapinfo heap;
    ostrstream totalStr( heapStr, sizeof heapStr);

    switch( farheapcheck() ) {
        case _HEAPEMPTY:
            strcpy(heapStr, "     No heap");
            total = -1;
            break;
        case _HEAPCORRUPT:
            strcpy(heapStr, "Heap corrupt");
            total = -2;
            break;
        case _HEAPOK:
            heap.ptr = NULL;
            while(farheapwalk(&heap) != _HEAPEND)
                if(!heap.in_use)
                    total += heap.size;
            totalStr << setw(12) << total << ends;
            break;
        }
    return(total);
    }


/*-- fine ----------------------------------------------------------------*/
```

D.9. ERR_FIND.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define BUFFERSIZE  200

char buffer[BUFFERSIZE];
char lastcodeline[BUFFERSIZE];

int iserrorline(char *b) {
    if(strncmp(b,"** ERROR",8)) return(0);
    return(1);
    }

int iscodeline(char *b) {
    while(isspace(*b)) b++;
    if(!isdigit(*b)) return(0);
    return(1);
    }

void main(int argc, char *argv[]) {

    FILE *fpin = stdin;
    FILE *fpout = stdout;

    if(argc>2) {
        if((fpin=fopen(argv[1],"r"))==NULL) {
            fprintf(stderr,"Can't open input file %s\n",argv[1]);
            exit(-1);
            }
        if((fpout=fopen(argv[2],"w"))==NULL) {
            fprintf(stderr,"Can't open output file %s\n",argv[2]);
            exit(-1);
            }
        }

    while(!feof(fpin)) {
        fgets(buffer,BUFFERSIZE-1,fpin);
        if(iscodeline(buffer)) {
            strncpy(lastcodeline,buffer,BUFFERSIZE);
            }
        if(iserrorline(buffer)) {
            fputs(lastcodeline,fpout);
            fputs(buffer,fpout);
            break;
            }
        }
    fclose(fpin);
    fclose(fpout);
    }
```

D.10. A!.BAT

```
@echo off
if exist %1.lst del %1.lst
if exist %1.obj del %1.obj
if exist %1.err del %1.err
asmhc11 %1;PROG=$%2
err_find %1.lst errors
REM Dateien der Länge 0 werden nicht kopiert...
copy errors %1.err > NUL
if exist errors del errors
if exist %1.err goto ende
copy %1.s19 %1.obj > NUL
:ende
del %1.s19
```

D.11. S!.BAT

```
@echo off
if not exist %1.lst goto err1
if not exist %1.obj goto err2
if exist %1.map del %1.map
mkmotmap %1.lst %1.map
if not exist %1.map goto err3
teste68 -m %1.obj -s %1.map
goto ende

:err1
echo Datei %1.lst nicht gefunden!
pause
goto ende

:err2
echo Datei %1.obj nicht gefunden!
pause
goto ende

:err3
echo Fehler beim Erzeugen der Symbol-Datei %1.map!
pause
goto ende

:ende
```

Selbständigkeitserklärung

Ich versichere, diese Diplomarbeit selbständig angefertigt und keine anderen als die angegebenen Hilfsmittel benutzt zu haben.

Leipzig, am 28. 2. 1994

Oliver Thamm