# Schnellstart mit dem HC12 Welcome Kit

Ein Tutorial für Einsteiger
von Andreas Dannenberg und Oliver Thamm

Der Motorola Mikrocontroller 68HC12 ist der neue "große Bruder" des weltweit und insbesondere in Deutschland sehr verbreiteten Controllers 68HC11. Im Gegensatz zu seinem 8-Bit Vorfahr besitzt der HC12 einen 16-Bit CPU-Kern mit erweitertem Befehlssatz, höherer Taktgeschwindigkeit (8 MHz Bustakt) und leistungsfähigeren Ein/Ausgabeschnittstellen. Für Anwenderprogramme stehen nunmehr bis zu 1 KB RAM und 4 KB EEPROM Speicherkapazität zur Verfügung. Vorhandenen Quellcode kann man leicht vom HC11 auf den HC12 übernehmen.

Für einen schnellen Einstieg in die neue Controllertechnologie bietet die Firma Elektronikladen Mikrocomputer ein HC12 Starterkit an. Das sogenannte HC12 Welcome Kit ermöglicht es, sofort mit dem Ausprobieren des Controllers und eigener Software zu beginnen, ohne langwierige Vorarbeiten im Hardwarebereich leisten zu müssen. Dieses Tutorial soll nun die Schritte von der Inbetriebnahme bis zum ersten Programm beschreiben.

# Überblick

Das HC12 Welcome Kit umfaßt neben dem Mikrocontroller HC12 und den obligatorisch zur Takterzeugung notwendigen Bauteilen auch einen Resetcontroller sowie einen Pegelwandler, der die beiden seriellen Schnittstellen des HC12 auf RS232-Standardpegel anpaßt und somit eine direkte Kommunikation mit anderen Rechnern ermöglicht. Alle Ein- und Ausgabeleitungen des Controllers sind über zwei jeweils 25x2-polige Stiftleisten zugänglich.

Ein wichtiger Bestandteil dieses Evaluation-Kits ist die schon in die oberen 2 KByte EEPROM des Controllers programmierte TwinPEEKs Monitorsoftware. Diese ermöglicht die einfache Programmierung des Kits über den Anschluß an die serielle Schnittstelle eines Host-PC mittels Terminalprogramm. Die Benutzung des Background-Debug-Mode (BDM) Anschlusses kann man vorerst umgehen, man vermeidet so den Kauf der dafür notwendigen Spezialsoftware.

# Vorbereitungen

Vor der ersten Inbetriebnahme sind die zwei mitgelieferten 50-poligen Stiftleisten auf das Controllermodul zu löten. Die Richtung kann man sich aussuchen. Weisen die Stiftleisten nach unten, kann das Board leicht auf eine andere Platine (Trägerplatine) aufgesteckt werden. Lötet man sie statt dessen nach oben ein, kann man die Signale z.B. über ein Flachbandkabel mit angecrimpten Pfostenverbinder abgreifen. Danach kann der ebenfalls mitgelieferte 4-polige Stromversorgungsstecker an das Kabel eines geeigneten Netzgerätes angelötet werden. Die Spannung sollte 5 Volt +/-10% betragen. Diese Versorgungsspannung sollte aus einem stabilisierten Netzgerät gewonnen werden. Finger weg von einfachen Steckernetzteilen, diese sind oftmals ungeregelt und daher völlig ungeeignet. Auf die richtige Polarität sollte man auch achten, Sünden werden hier umgehend bestraft. Über die genaue Pinbelegung kann man sich im Zweifelsfalle jederzeit im Hardwarehandbuch des Kits informieren.

Um Kurzschlüsse und die damit eventuell verbundene Zerstörung des Boards durch Aufliegen auf elektrisch leitfähigen Materialien zu vermeiden, ist es ratsam, z.B. mittels dreier, gleich langer M3-Schrauben, welche von oben in die Platine geschraubt werden, einen gewissen Abstand zum Untergrund zu schaffen. Es sollte weiterhin eine Kontrolle der Jumpereinstellungen durchgeführt werden, damit das Board im sog. "Normal Single Chip"-Modus startet (JP1: 1-2, JP2: 1-2, JP3: 2-3). Dieser ist für die ordnungsgemäße Arbeit mit der integrierten Monitorsoftware nötig.

Jetzt kann die erste serielle Schnittstelle des Controllerboards (beschriftet mit "SER0") mit einem RS232-Anschluß des (ausgeschalteten) Host-PC verbunden werden. Das dazu notwendige serielle, gekreuzte Verbindungskabel liegt dem Kit bei. Nach dem Anschluß der Stromversorgung an das Kit und dem Einschalten des PCs muß nun noch ein Terminalprogramm geladen werden. Der Einfachheit halber sei hier das zu jedem Windows 3.X mitgelieferte Programm TERMINAL.EXE empfohlen, welches auch unter Windows 95/98 lauffähig ist und sehr einfach zu bedienen ist. Wer ein anderes Terminalprogramm bevorzugt, findet die im Folgenden genannten Einstellungen u.U. mit anderen Bezeichnungen und an anderer Stelle.

Zunächst ist die richtige serielle Schnittstelle auszuwählen und die vom Monitorprogramm erwarteten seriellen Übertragungsparameter einzustellen (9600 Baud, 8 Datenbits, 1 Stoppbit, keine Parität im Menü Einstellungen/Datenübertragung). Weiterhin ist eine evtl. eingeschaltete Umlautwandlung (Menü Einstellungen/Terminal-Einstellungen) auszuschalten.

Für das ordnungsgemäße Laden von Programmdateien auf das Controllerboard sind noch die Einstellungen für den S-Record Download (Menübefehl "Textdateiübertragung") anzupassen. Dies ist bedingt durch die relativ langsame Programmiergeschwindigkeit des im Controller integrierten EEPROMs. Wenn die ersten Programme nur in den RAM-Speicher übertragen werden sollen, ist diese Einstellung nicht so wichtig. Beim Download in den EEPROM muß aber ein bestimmtes Timing gewährleistet sein. Nach jeder verarbeiteten S-Recordzeile sendet das Monitorprogramm ein Quittungszeichen "*"

zurück. Das Terminal muß einfach auf dieses Zeichen warten, bevor die nächste Zeile übertragen wird. Bei TERMINAL.EXE wählt man im Menü Einstellungen/Textübertragung die Option "Zeilenweise Übertragung" und trägt "*" als "Sendeaufforderung" ein.

## Prompte Bedienung

Das Controllerboard müßte sich nun mit einem mit "TwinPEEKs Ver1.2...." beginnenden, mehrzeiligen Willkommenstext, gefolgt vom Eingabeprompt ";->" melden. Durch Betätigen der <ESC> Taste kann man einen Soft-Reset (Warmstart) auslösen, die Meldung wird dann erneut ausgegeben. Jetzt können schon erste Monitorkommandos ausprobiert werden, welche jeweils durch drücken der <ENTER> Taste ausgeführt werden. Die Eingabe von "H" bringt beispielsweise eine kurze Befehlsübersicht ans Tageslicht. Die erste Kommunikation ist geschafft!

Hier ist eine kurze Erklärung zu den Monitorbefehlen:

| Befehl | Erläuterung |
|---|---|
| `d [[adr1] adr2]` | zeigt den Speicherbereich von adr1 bis adr2 als Hex-Dump an |
| `e [addr]` | dient der Eingabe von Bytes ab Adresse addr, beenden mit "Q" |
| `f adr1 adr2 dd` | füllt den Bereich zwischen adr1 und adr2 mit dem Byte dd |
| `g [addr]` | führt ein an der Adresse addr befindliches Programm aus |
| `h` | stellt kurze Befehlsübersicht dar |
| `l [offs]` | lädt eine .S19-Textdatei mit dem Offset offs |
| `m adr1 adr2 adr3` | kopiert den Bereich von adr1 bis adr2 an die Adresse adr3 |

Alle Adreßeingaben erfolgen Hexadezimal, Parameter in eckigen Klammern sind optional. Befehle als auch Adreßangaben können wahlweise in Groß- oder Kleinbuchstaben erfolgen.

Wichtig in diesem Zusammenhang ist weiterhin die Belegung des 64 KByte, vom Controller direkt adressierbaren Speicherbereiches (normale Betriebsart: "Single Chip Mode"):

```
$0000-$01FF    Registerblock
$0800-$0AFF    RAM frei für Anwenderprogramm (768 Byte)
$0B00-$0BB7    RAM für Monitor Variablen und Stack
$0BB8-$0BFF    RAM für umgeleitete Interruptvektoren
$F000-$F7FF    EEPROM frei für Anwenderprogramm (2 KByte)
$F800-$FFFF    EEPROM mit Monitorcode (geschützt, 2 KByte).
```

## Leuchtendes Beispiel

Nun können wir uns an den Anschluß und die Ansteuerung einer Leuchtdiode "wagen". Die folgenden Ausführungen beziehen sich auf das (willkürlich gewählte) Bit 7 des I/O-Ports H (PH7). Der Aufbau dieser kleinen Testschaltung erfolgt praktischerweise direkt auf den Lötkontakten eines 50-poligen Pfostenverbinders, der auf die zuvor eingelöteten Stiftleisten gesteckt wird. So vermeidet man es, direkt an den Kontaktstiften löten zu müssen.

Die Kathode der LED ist an den Pin 35 (PH7) des mit ST6 bezeichneten Steckverbinders anzuschließen, die Anode über einen geeigneten Vorwiderstand an Pin 50 (VCC). Dessen Wert sollte, um die zulässige Strombelastung des I/O-Portes nicht zu überschreiten, etwa 1 kOhm betragen. Wegen Ihrer höheren Leuchtkraft wird in diesem Fall eine Low-Current LED empfohlen. Der Einsatz einer normalen Leuchtdiode ist auch möglich, nur leuchtet diese dann mit geringer Intensität.

Nach dem Aufstecken des Pfostensteckers auf das Controllerboard kann man die Funktionsfähigkeit der LED durch die direkte Manipulation im Registerblock des Controllers überprüfen. Die dafür wichtigen Speicherzellen liegen bei $0025, dem Datenrichtungsregister des Portes H (kurz: DDRH) und bei $0024, dem dazugehörigen Datenregister

(kurz: PORTH). Das Datenrichtungsregister entscheidet, welche Leitungen des I/O-Ports als Ausgang (die entsprechende Bitposition ist "1") oder als Eingang (entsprechende Bitposition ist "0") verwendet werden. Standardmäßig sind alle Pins als Eingang programmiert, d.h. an Adresse $0025 ist eine $00 eingetragen. Dies können wir leicht mittels Monitorprogramm überprüfen: die Eingabe des Kommandos "d 0025" mittels Terminalprogramm liefert "0025:  00 00 00...". Nun wollen wir zuerst das gesamte Port H als Ausgang konfigurieren, also den Wert $FF an die Adresse $0025 eintragen.

Dies geschieht durch die Eingabe von:

```
E 0025<ENTER>
FF<ENTER>
Q<ENTER>
```

Daraufhin beginnt die LED zu leuchten. Was ist passiert? Im Datenregister PORTH steht nach Reset immer $00 ("Low"). Auch alle Bits im Datenrichtungsregister DDRH sind nach Reset auf Null und das bedeutet "Eingang". Nach dem Umschalten auf "Ausgang" wird der Low-Pegel auf Pin PH7 ausgegeben und sorgt damit für Stromfluß durch LED und Vorwiderstand gen Betriebsspannung. Durch Eintragen von $80 (entspricht binär %1000 0000) in PORTH (Adresse $0024) können wir die Portleitung PH7 auf HIGH-Potential legen und damit die LED ausschalten. Die Eingabe muß analog zu oben lauten:

```
E 0024<ENTER>
80<ENTER>
Q<ENTER>
```

Die LED sollte nun nicht mehr leuchten.

## Blinkversuche

Die anfänglichen Hürden haben wir erfolgreich gemeistert, nun können wir uns an das erste "richtige" Programm wagen. Die Aufgabe

bestehe darin, unsere eben angeschlossene LED im Sekundentakt blinken zu lassen.

Zur Programmentwicklung benötigen wir den auf der CD-ROM (im Lieferumfang des Kits) enthaltenen Assembler. Die Installation der Software ist einfach, es werden einfach alle Dateien aus dem Pfad \AS von der CD in ein Unterverzeichnis auf der Festplatte kopiert. Der vom Autor Alfred Arnold stammende Assembler gehört in die Kategorie "kostenlos benutzbar". Er unterstützt eine ganze Reihe unterschiedlicher CPUs, u.a. den von uns eingesetzten HC12. Zu Beginn des Quelltextes muß man den Assembler mittels zweier Anweisungen (cpu, padding) auf den HC12 einstellen. Hier ist das Programmlisting, welches mit einem beliebigen Texteditor abgetippt und als "BLINK.A" im "AS"-Verzeichnis gespeichert wird:

```
;===========================================================================
; Name: BLINK.A
; Ver.: 1.0
; Fkt.: Assembler Demo - läßt LED an PH7 blinken (etwa 0.5Hz Blinkfrequ.)
; Auth: Andreas Dannenberg, 12. Juli 1999
; Bem.: übersetzen mit "A12 BLINK"
;===========================================================================

        cpu 68hc12      ; Assemblerspezifische Anweisungen für
        padding off     ; das Erzeugen von HC12-Code

        org  $0800      ; Programm im Anwenderspeicher ablegen

        lds  #$0c00     ; Stack Zeiger initialisieren

        ldaa #$ff       ; alle Pins auf Ausgabe stellen
        staa $0025      ; ins Datenrichtungsregister von Port H eintragen

loop    ldaa #$80       ; LED ausschalten (Pin PH7 auf HIGH-Potential)
        staa $0024      ; Wert ins Port H Datenregister (PORTH) eintragen
        bsr  wait       ; Unterprogramm Zeitschleife aufrufen
        ldaa #$00       ; LED einschalten (Pin PH7 auf LOW-Potential)
        staa $0024      ; Wert ins Port H Datenregister (PORTH) eintragen
        bsr  wait       ; wieder einen Moment warten...
        bra  loop       ; zur Hauptschleife verzweigen

wait    ldx  #$2000     ; Unterprogramm wartet etwa eine Sekunde
        ldaa #$00       ; verschachtelte Schleife nötig wegen
                        ; hoher Rechengeschwindigkeit
wloop   dbne a,wloop    ; innere Schleife 256 mal durchlaufen
        dbne x,wloop    ; äußere Schleife 8192 mal (=$2000)
        rts             ; danach Rückkehr ins Hauptprogramm
```

Danach ist der Assemblerquelltext in eine sogenannte S19-Datei zu übersetzen. Dazu wechselt man in der MS-DOS Ebene in das "AS"-Verzeichnis und ruft den Assembler auf mit dem Kommando "A12 BLINK"  Die entstandene S19-Datei enthält den Hex-Dump des fertigen Maschinenprogrammes. Für weitere Informationen zum S19 Dateiformat sei ein Blick in das dem Kit beiliegende TwinPEEKs-Handbuch empfohlen.

Bevor wir das Programm starten können, müssen wir es zunächst vom PC zum Controllerboard übertragen. Nach dem Start des PC-seitigen Terminalprogrammes ist das Monitorkommando "L" (für Load) einzugeben. Nun ist der Controller empfangsbereit für das Programm. Die übersetzte S19-Datei ist nun mit der Funktion "Textdatei Senden" des Terminalprogrammes zu senden. Nach erfolgreicher Übertragung meldet sich das Monitorprogramm wieder mit dem Eingabeprompt. Nun können wir das Programm mit dem Kommando "G 0800" ausführen. Wenn nun die LED blinkt, liegt der erste Abschnitt erfolgreich hinter uns. Weiteren Programmierversuchen sollte nun nichts mehr im Wege stehen!

Ausführliche Informationen zu den Möglichkeiten des Controllers und dessen umfangreichen Befehlssatz halten die mitgelieferten Datenblattdateien im PDF-Format bereit. Wichtige und oft benötigte Teile, z.B. der Registerblock im HC812A4 Datenblatt und die Befehlsübersicht im CPU12 Reference Manual sollten am Besten gleich ausgedruckt werden, damit man die Informationen stets zur Hand hat.

## Tips & Tricks

Zur Vermeidung des ständigen Unterbrechens der Stromversorgung bei Programmfehlern oder der Rückkehr zum Monitorprogramm ist es empfehlenswert, mittels eines 2-poligen Steckverbinders und einem ca. 20 cm langen, zweipoligen Kabel einen Resettaster am Jumper JP4 anzuschließen.

Ein Programm in den RAM zu laden und auszuführen ist offenbar kein Problem. Wie aber kann man ein Programm so speichern, daß es nach dem Einschalten sofort zu arbeiten beginnt, ohne daß sich zuvor der Monitor "dazwischen mogelt"? Das HC12 Welcome Kit bietet hierzu eine Autostartfunktion. Voraussetzung ist, daß man das Programm in den EEPROM ab Adresse $F000 lädt. Der Autostart funktioniert dann so: Nach dem Anschließen der Versorgungsspannung oder nach einem Reset prüft das Monitorprogramm, ob die Pins 35 (PH7) und 37 (PAD0) des Steckverbinders ST6 miteinander verbunden sind. Ist das der Fall, wird das an Adresse $F000 im EEPROM gespeicherte Programm sofort ausgeführt.

Im EEPROM steht standardmäßig ein 2 KByte großer Speicherbereich zur Verfügung ($F000-$F7FF). Die oberen 2 KByte des EEPROM werden vom Monitorprogramm belegt (siehe Speicherbelegungstabelle). Um diese Option auszuprobieren, löten wir über ein kurzes Stück zweiadriges Kabel einen kleinen Kippschalter an die benachbarten Pins 35 (an welchem schon die LED angeschlossen ist) und 37 an. Danach ändern wir in unserem Quelltext BLINK.A die Zeile "org $0800" in "org $f000" und übersetzen das Listing durch die Eingabe von "A12 BLINK" am MS-DOS Prompt neu. Dadurch wird der Programmanfang an die Adresse $F000 gelegt - der Autostart-Einsprungpunkt. Anschließend ist das entstandene .S19-File wieder mittels der "Textdatei Senden"-Funktion des Terminalprogrammes an das Controllerboard zu übertragen. Durch Einschalten unseres soeben angebrachten Autostart-Schalters und Auslösen eines Resets bzw. Neuanschluß der Stromversorgung an unser HC12 Welcome Kit, fängt nun unser Programm völlig "selbständig" an zu laufen und die LED beginnt zu blinken. Die Rückkehr zur Monitorebene kann nur durch Auftrennen der Autostartverbindung und Betätigen des Reset-Tasters erfolgen.

## Kleine Nachtmusik

Mit diesem Handwerkszeug ausgestattet, möchten wir uns noch an ein etwas komplexeres Programmierprojekt wagen, obwohl dieses natürlich auch nicht das gesamte Leistungspotential unseres Controllers repräsentieren kann. Der Controller soll mit der Ausgabe einer Melodie beschäftigt werden.

Dazu sind erst einige kleine hardwareseitige Vorbereitungen nötig. Für unser Projekt benötigen wir einen kleinen Lautsprecher und einen Widerstand von etwa 500 Ohm. Einen Anschluß des Minilautsprechers löten wir über einen Vorwiderstand an Pin 50 (VCC) des Steckverbinders ST6 an. Da wir aber nun an bestimmte Portpins, genauer gesagt die Timer-Portpins gebunden sind, kommen wir nicht daran vorbei, auch den zweiten 50-poligen Steckverbinder zu verwenden. Mit dessen Pin 44 (Timer Port 0 = PT0) ist wieder mittels eines Pfostensteckverbinders die andere Klemme des Lautsprechers zu verbinden.

Um das etwas umfangreichere Programmlisting nicht abtippen zu müssen, liegt es auch auf dem Webserver des Elektronikladen (http://www.elektronikladen.de) zum Download bereit.

## Zählen und zählen lassen

Der Controller besitzt ein kontinuierlich arbeitendes Zählerregister. Es ist 16-Bit breit, der Zählumfang ergibt sich somit zu 0 bis 65535. In gleichbleibenden Abständen (einstellbar von 125 ns bis 4 µs) wird der Zähler weiter getaktet. Nach Erreichen des höchsten Wertes geht es mit Null von vorn los. Der Zählerstand ist jederzeit über das zwei Byte umfassende "Timer/Counter Register" (TCNT) abrufbar. Die Zählfrequenz wird mit drei Bits im "Timer Interrupt Mask 2 Register" (TMSK2) festgelegt - Einzelheiten siehe HC12-Datenblatt.

Für die gezielte Ausgabe bestimmter Frequenzen möchten wir das Interruptsystem unseres Controllers im Zusammenspiel mit der "Output-Compare"-Funktion benutzen. Unmittelbares Ziel ist es dabei, eine

Portleitung in bestimmten Abständen (Periodendauer) ein- bzw. auszuschalten.

Ein eleganter Weg besteht in der Nutzung einer von acht mit dem Timersystem verbundenen Portleitungen (PORTT). Diese können automatisch zu speziellen Ereignissen (Output-Compare Interrupts) im Controllerinneren ein-, aus- oder umgeschaltet werden.

Und so läuft ein Output-Compare Ereignis ab: Für jede der acht Leitungen des Port T ist ein 16 Bit breites, genanntes Steuerregister vorhanden (TC0 .. TC7). Eine interne Logik vergleicht ständig den Inhalt dieser Register mit dem Inhalt des freilaufenden Zählers TCNT. Wenn nun der Wert eines dieser Register mit dem Timer übereinstimmt, spricht man von einem Output-Compare Ereignis. Als Reaktion können zwei Aktionen stattfinden. Zunächst wird eine Programmunterbrechung, also ein Interrupt angemeldet. Zudem kann man die zum Ereignis gehörige Portleitung (ein-, aus-, um-) schalten lassen. Das geschieht "hardware-automatisch" und zudem genau im Augenblick der Übereinstimmung von Output-Compare Register und Zähler TCNT.

Und wie ist das mit den Interrupts? Ist die Interruptlogik freigegeben (dazu später mehr), rettet die CPU automatisch den Inhalt sämtlicher CPU-Register und verzweigt zur Interruptserviceroutine. An welcher Adresse sich diese vom Benutzer verfaßte Interruptbehandlung befindet, trägt man in eine Interrupt-Vektortabelle ein. Diese befindet sich ganz am Ende des HC12 Programmspeichers und enthält für jeden HC12 Interrupt einen zwei Byte umfassenden Eintrag - die Startadresse der Interrupt.

Die aufgerufene Interruptserviceroutine sollte den Code enthalten, der die planmäßige, erneute Auslösung eines Interrupts veranlaßt. In diesem Zusammenhang ist auch das Löschen des entsprechenden Interruptflags obligatorisch. Man erreicht das Löschen eines Interruptflags interessanterweise durch Beschreiben der Bitposition mit einer Eins (merken!). Man sollte dabei auf eine einfache STAA- oder MOVB-Anweisung zurückgreifen, eine BSET-Anweisung hingegen kann hier große Probleme verursachen! Nach der Rückkehr aus der Interruptserviceroutine per RTI-Anweisung restauriert die CPU die eingangs

gemerkten CPU-Register und führt das Hauptprogramm an der Stelle fort, an der die Unterbrechung stattfand. Voila!

## Quelltext-Kompositionen

Dieses System der Timerinterrupts wollen wir jetzt zur Ausgabe bestimmter Tonfrequenzen benutzen. Die Periodendauer T einer Frequenz f läßt sich über die Formel T=1/f bestimmen und daraus die Anzahl der Timerschritte ableiten, nach denen unsere Portleitung umzuschalten ist.

Im ersten Teil des Quelltextes werden per include-Anweisung des Assemblers die Registerdefinitionen des HC12 eingefügt. Das erspart uns das ständige Hantieren mit den hexadezimalen Adressen der Steuerregister. Diese Definitionen können in der Includedatei "reghc12.inc" eingesehen werden und stimmen mit den im Datenblatt des Controllers aufgeführten Bezeichnungen überein.

Das Programm wird im EEPROM (ab Adresse $F000) des Controllers abgelegt. Zusätzlich wird aber noch eine Merkzelle für die Zwischenspeicherung einer Variable benötigt. Deshalb sind im Listing zwei ORG Anweisungen enthalten. Die erste ("org $0800") sorgt dafür, daß im RAM-Bereich Speicher reserviert wird, die zweite ist für die Startadresse des Codes im EEPROM zuständig. Die Reservierung des Speichers geschieht mittels "ds.w"-Pseudoanweisung, was soviel heißt wie "define space word". Es wird also ein Wort (zwei Byte) im RAM reserviert und mit dem Label "BuzzPeriod" darauf verwiesen. Dieser Speicherplatz wird zur Ablage der Periodendauer der auszugebenden Tonfrequenz benutzt.

## Pseudovektoren

Die Interruptvektoren liegen normalerweise am Ende des Speichers. Diese Stelle belegt auf dem HC12 Welcome Kit jedoch das Monitorprogramm. Um das Interruptsystem dennoch nutzen zu können, leitet TwinPEEKs alle Vektoren in den RAM um. Dort werden nun 3 Byte in eine "Pseudo"-Interruptvektortabelle eingetragen. Um einen

Zeiger für die Interruptroutine des Output-Compare 0 Registers einzurichten, wird der Opcode des JMP-Befehls ($06) an die Adresse $0BE8 und der Zeiger auf die Routine an die Adressen $0BE9 (High-Byte) und $0BEA (Low-Byte) geschrieben. Die Adressen der anderen Interruptvektoren sind im TwinPEEKs-Handbuch dokumentiert.

Im "Timer Interrupt Mask 2 Register" (TMSK2) wird nun der Vorteiler des Hauptzählers auf 4 eingestellt. Bei einem Systemtakt von 8 MHz bedeutet das eine Zeitdauer von 0,5 µs zwischen den Zählschritten. Nach dem Einschalten des Timersystems durch Setzen des Bit 7 im "Timer System Control Register" (TSCR) und der Konfiguration des Portpins PT0 als Output-Compare im "Timer Input Capture/Output Compare Select Register" (TIOS), wird durch einen Eintrag im "Timer Control 2 Register" (TCTL2) der Ausgabepin zunächst vom Timersystem abgekoppelt. Dadurch wird die (unerwünschte) Ausgabe von Tonfrequenzen unterbunden, sozusagen eine Mute-Funktion.

Damit vom Controller Interrupts ausgeführt werden können und unsere Serviceroutine angesprungen wird, ist sowohl die lokale Interruptfreigabe des OC0 Registers durch Setzen des Bits 1 im "Timer Interrupt Mask 1 Register" (TMSK1), als auch die globale Interruptfreigabe mittels CLI Anweisung zu aktivieren.

Von diesem Moment an wird bei jeder Übereinstimmung des TC0 Registers mit dem Timer-Register TCNT die Interruptserviceroutine isrOC0 angesprungen. Als erstes müssen wir dafür sorgen, daß ein neuer Output-Compare Wert eingetragen wird. Diesen neuen Wert errechnen wir, indem wir zum augenblicklichen Stand des Output-Compare Registers eine Konstante addieren. Diese Konstante entspricht der gewünschten Periodendauer. Sie stammt aus der im Hauptprogramm reservierten Speicherzelle BuzzPeriod. Das Ergebnis wird in das Output-Compare Register zurückgeschrieben. Weiterhin wird durch das Schreiben einer Eins auf das "Timer Interrupt Flag 1 Register" (TFLG1) das OC0 Interruptflag gelöscht.

Das Hauptprogramm besteht aus einer Schleife, welche nacheinander die Notenwerte unseres Songs einliest und auswertet. Jeder Notenwert wird dekodiert, um Frequenz und Tondauer (mit Hilfe der beiden Tabellen freqTBL und durTBL) zu ermitteln. Die Frequenz wird danach

in eine Periodendauer umgerechnet und diese über die Merkzelle BuzzPeriod an die isrOC0-Routine übergeben. Dabei ist zu beachten, das nur die halbe Periodendauer der tatsächlich auszugebenden Frequenz ermittelt wird, da zur Erzeugung einer Frequenz ja innerhalb deren Periodendauer zweimal der Pegel gewechselt werden muß.

Anschließend wird die Tonausgabe, soweit es sich nicht gerade um eine Pause handelt, durch das Setzen des Bit 1 im "Timer Control 2 Register" (TCTL2) eingeschaltet. Das bewirkt, daß die im Controller integrierte Logik bei jedem Auslösen des TC0 Interrupts den Pegel unserer Portleitung invertiert (Toggle-Modus). Die dadurch ausgegebene Rechteckimpulsfolge wird als Ton hörbar.

Nach Ablauf der aus der durTBL-Tabelle ermittelten Notendauer wird der Ton wieder abgeschaltet (Löschen des Bit 1 im TCTL2 Register). Eine zusätzliche kurze Pause nach jedem Ton gibt der Melodie etwas mehr Rhythmus. Anschließend erfolgt eine Verzweigung zurück zum Beginn des Hauptprogrammes. Es werden solange weitere Notenwerte bearbeitet, bis das Programm auf die Note mit der Ende-Markierung trifft ($80 = Bit 7 gesetzt).

## Noten aus der Tabelle

Die Noten des zu spielenden Liedes sind am Programmende unter der Marke song abgelegt. Um beim Komponieren nicht ständig mit den Zahlenwerten der Tabellenoffsets hantieren zu müssen, wurden diese durch symbolische Konstanten mittels EQU-Pseudoanweisung ersetzt. Beim Übersetzen ersetzt der Assembler nun alle Zeichenfolgen, welche links einer EQU-Anweisung stehen, durch die der entsprechenden rechten Seite. So wird beispielsweise ein mit "_AIS" bezeichneter Notenwert durch die Zahl 11 ersetzt, welche dann auf die Frequenz 932 Hz in der freqTBL-Tabelle verweist.

Die Definition der Notenwerte wird durch die dc.b Anweisung realisiert welche Speicherplatz für ein Byte bereitstellt. Wird eine Note ohne weitere Angaben eingetragen, erfolgt die Tonausgabe mit der Länge einer 1/4-Note in der oberen der beiden vorgesehenen Oktaven. Zur Realisierung abweichender Tonlängen bzw. Oktaven sind die im

Listing definierten Symbolischen Konstanten _1_8, _3_8 und _1_2 bzw. _DEEP durch einfache Addition zum Notenwert hinzuzufügen. Beispielsweise ergibt die Zeile

```
dc.b _A + _1_2 + _DEEP
```

die Ausgabe eines langen Kammertones A. Das Songende wird durch die Marke _STOP gekennzeichnet.

Nach dem Übersetzen des Quelltextes, Senden an unser HC12 Welcome Kit und Start mittels Autostart-Funktion (oder der Eingabe von "G F000") werden wir - jetzt fast schon HC12 Experten - mit einer netten Melodie verwöhnt.

## Viel Spaß beim Komponieren!

(mailto:leipzig@elektronikladen.de)

```
;=====================================================================
; Name: MELODIE.A
; Ver.: 1.01
; Fkt.: Assembler Demo - gibt Melodie über Lautsprecher an PT0 aus
; Auth: Andreas Dannenberg, Oliver Thamm
; Bem.: übersetzen mit "A12 MELODIE"
;=====================================================================

                cpu 68hc12              ; Assemblerspezifische Anweisungen
                padding off             ; für das Erzeugen von HC12-Code

                include "reghc12.inc"   ; Registerdefinitionen einbinden

                org   $0800             ; Variablenbereich
BuzzPeriod      ds.w  1                 ; Merkzelle für Periodendauer
                                        ; akt. Note

                org   $f000             ; Programm im EEPROM unterbringen

;---------------------------------------------------------------------
; Ausführung div. Initialisierungen
;---------------------------------------------------------------------
                lds   #$0b00            ; Stack Pointer initialisieren

                ldaa  #$06              ; JMP Opcode laden und in Interrupt-
                staa  $0be8             ; Vektortabelle schreiben (TC0)
                ldd   #isrOC0           ; Zeiger auf Int. Service-Routine
                std   $0be9             ; in Vektortabelle schreiben

                movb  #$32,TMSK2        ; TimerPrescaler=4 (tick=0.5µs)
                movb  #$80,TSCR         ; Timer System einschalten
                bset  TIOS,$01          ; PT0 auf Output Compare stellen
```

```
                bclr  TCTL2,$03          ; Timer v. PT0 Pin trennen (Ton aus)
                bset  TMSK1,$01          ; lokalen Interrupt für OC0 freig.
                cli                      ; globale Interruptfreigabe

                ldy   #song              ; Zeiger auf 1. Notenwert setzen
;-------------------------------------------------------------------------
; Hauptprogramm-Schleife (spielt Notenwerte aus Tabelle song)
;-------------------------------------------------------------------------
playLoop        ldab  0,y                ; nächste Note aus Songtabelle holen
                cmpb  #$80               ; Songende schon ereicht?
                blo   nextTone           ; wenn nicht, weiterspielen

ende            bra   ende               ; wenn ja:
                                         ; Programmende=Endlosschleife

nextTone        pshb                     ; Notenwert auf Stack sichern
                andb  #%00001111         ; Notenbits 0..3 isolieren
                clra                     ; oberes Byte des Akkus D löschen
                lsld                     ; mit 2 multiplizieren
                addd  #freqTBL           ; Offset der Frequenztabelle
                                         ; addieren
                tfr   d,x                ; ins Indexregister laden
                ldx   0,x                ; und Frequenz aus Tabelle holen

                pulb                     ; Notenwert vom Stack zurückholen
                cpx   #0                 ; Frequenz auf 0 prüfen
                beq   pause              ; Wenn Note=0, Pause einlegen
                andb  #%00010000         ; Note auf tiefe Oktave prüfen
                                         ; (Notenbit 4)
                beq   hoheOkt            ; wenn nicht, Frequenz nicht
                                         ; halbieren

                tfr   x,d                ; x => d Register
                lsrd                     ; Frequenz halbieren
                tfr   d,x                ; d => x Register

hoheOkt         pshx                     ; Frequenz auf Stack retten
                ldd   #62500             ; 1,000,000 / 16
                idiv                     ; x = d/x, Rest in Akku D
                xgdx                     ; d <=> x
                lsld
                lsld                     ; Ergebnis der Division mit
                lsld                     ; 16 multiplizieren
                lsld
                xgdx                     ; d <=> x
                stx   BuzzPeriod         ; in der Periodendauerzelle merken
                pulx                     ; Frequenz vom Stack zurückholen
                lsld
                lsld
                lsld                     ; Rest obiger Division
                lsld                     ; mit 16 multiplizieren
                idiv                     ; nochmal durch Frequenz teilen
                tfr   x,d                ; und in Akku D übertragen
                orab  BuzzPeriod+1       ; die letzten 4 Bit der Gesamtdiv.
                stab  BuzzPeriod+1       ; auch in die Periodendauerzelle
                                         ; übertragen
                bset  TCTL2,$01          ; OC0 action: toggle PT0 (Ton an)
```

```
pause           ldab  0,y               ; Notenwert nocheinmal holen
                andb  #%01100000        ; Tondauer-Bits isolieren
                lsrb                    ; durch 16 Teilen, um daraus den
                lsrb                    ; Offset der Tonlänge (word) in
                lsrb                    ; der Tabelle zu bestimmen
                lsrb
                clra                    ; oberes Halbbyte v. Akku D löschen
                addd  #durTBL           ; Offset der Tondauer-Tabelle add.
                tfr   d,x               ; Wert ins Indexregister laden
                ldx   0,x               ; Tonlängenwert aus Tabelle holen

                ldaa  #$ff              ; verschachtelte Schleife wartet,
ploop           dbne  a,ploop           ; bis Note zu Ende gespielt ist
                dbne  x,ploop

                bclr  TCTL2,$03         ; Timer v. PT0 Pin trennen (Ton aus)

                ldx   #1300             ; bis zum Spielen der nächsten Note
                ldaa  #$ff              ; noch einen Moment warten
ploop2          dbne  a,ploop2          ; verschachtelte Schleife
                dbne  x,ploop2

                iny                     ; Notenzeiger um 1 erhöhen
                bra   playLoop          ; und nächste Note spielen

;-------------------------------------------------------------------------
; Interrupt-Service-Routine (ISR) für die Output Compare (OC) 0 Funktion
;-------------------------------------------------------------------------
isrOC0          movb  #$01,TFLG1        ; OC0 Interrupt-Flag durch
                                        ; Beschreiben löschen
                ldd   TC0               ; alten Zählwert aus Timer-Register
                                        ; holen
                addd  BuzzPeriod        ; dazu Periodendauer der aktuellen
                                        ; Frequenz addieren
                std   TC0               ; und in OC0-Register zurückschr.
                rti                     ; Rückkehr zur Unterbrechungsstelle
                                        ; im Programm

;-------------------------------------------------------------------------
; Tabelle zur Umwandlung eines Notenwertes (Zahl 0..12) in eine Frequenz
;-------------------------------------------------------------------------

freqTBL         dc.w    0               ; pause
                dc.w  523               ; c
                dc.w  554               ; cis
                dc.w  587               ; d
                dc.w  622               ; dis
                dc.w  659               ; e
                dc.w  698               ; f
                dc.w  740               ; fis
                dc.w  784               ; g
                dc.w  830               ; gis
                dc.w  880               ; a
                dc.w  932               ; ais
                dc.w  987               ; h
```

```
;-------------------------------------------------------------------------
; Tabelle zur Umwandlung der Notendauer in einen Zeitschleifen-Wert
;-------------------------------------------------------------------------
durTBL          dc.w    1200            ; 1/4
                dc.w     600            ; 1/8
                dc.w    2400            ; 1/2
                dc.w    1800            ; 3/8

;-------------------------------------------------------------------------
; Notendefinitionen (gesamter Notenwert passt in einzelnes Byte)
;
; Die Erzeugung eines Musikstückes erfolgt durch Addition der gewünschten
; Symbolischen Konstanten und Ablage im Speicher per "dc.b"-Anweisung.
; Das Ende des Liedes ist mit der Symbolischen Marke "_STOP" zu kennzeichn.
;
; Beispiel: song        dc.b    _A + _DEEP      ; Kammerton A, 1/4-Note
;                       dc.b    _PP + _1_2      ; Pause, Länge 1/2-Note
;                       dc.b    _GIS            ; G#, hohe Oktave
;                       dc.b    _STOP           ; Ende
;-------------------------------------------------------------------------
_PP     equ     0       ; Code für Pause (Ton stummgeschaltet)
_C      equ     1       ; Werte der 12 Töne einer Oktave
_CIS    equ     2
_D      equ     3
_DIS    equ     4
_E      equ     5
_F      equ     6
_FIS    equ     7
_G      equ     8
_GIS    equ     9
_A      equ     10
_AIS    equ     11
_H      equ     12

_HIGH   equ     $00     ; Oktaven-Codes (Bit 4), hohe Oktave ist voreingest.
_DEEP   equ     $10     ; Tiefe Oktave = Bit 4 gesetzt

_1_8    equ     $20     ; Tondauer-Codes, Bits 5 und 6
_1_4    equ     $00     ; 1/4-Noten sind voreingestellt
_3_8    equ     $60
_1_2    equ     $40

_STOP   equ     $80     ; Marke für Songende

;-------------------------------------------------------------------------
; Demosong: "Love Story Theme"
;-------------------------------------------------------------------------
song            dc.b    _AIS
                dc.b    _D
                dc.b    _D
                dc.b    _AIS
                dc.b    _AIS    +_1_2
                dc.b    _PP
                dc.b    _D
                dc.b    _D
                dc.b    _AIS
                dc.b    _AIS
```

```
                dc.b    _D
                dc.b    _DIS
                dc.b    _D
                dc.b    _C
                dc.b    _C
                dc.b    _C
                dc.b    _A
                dc.b    _A      +_1_2
                dc.b    _PP
                dc.b    _C
                dc.b    _C
                dc.b    _A
                dc.b    _A
                dc.b    _C
                dc.b    _D
                dc.b    _C
                dc.b    _AIS    +_DEEP
                dc.b    _AIS    +_DEEP
                dc.b    _AIS    +_DEEP
                dc.b    _G
                dc.b    _G      +_1_2
                dc.b    _PP
                dc.b    _AIS    +_DEEP
                dc.b    _AIS    +_DEEP
                dc.b    _G
                dc.b    _G
                dc.b    _AIS    +_DEEP
                dc.b    _C
                dc.b    _AIS    +_DEEP
                dc.b    _A      +_DEEP
                dc.b    _A      +_DEEP
                dc.b    _A      +_DEEP
                dc.b    _FIS
                dc.b    _FIS    +_1_2
                dc.b    _PP
                dc.b    _G      +_DEEP
                dc.b    _A      +_DEEP
                dc.b    _DIS    +_DEEP
                dc.b    _D      +_1_2   +_DEEP
                dc.b    _STOP                   ; Marke für Songende

;-------------------------------------------------------------------------
```