



SYSTECH J.Schnyder GmbH

Schlifweg 30
CH-4106 Therwil
Telefon 091 827 15 87

EBS08

Einfaches Betriebs-System für Motorola HC08 Microcontroller

für
HC08AB32
HC08Qxy

V 0.5 P&E

Beschrieb

Inhalt

Einleitung	3
Die Entwicklungs-Umgebung	3
Das Grundsystem	3
Zusatz-Module	3
Beschreibung des Systems	4
Steuer-File	4
TIC-Modul	4
INI-Routine	5
TASK-Manager	5
Vektor-Block	6
EBS08-Timing	7
Timing des Beispiel-Programms	7
Namens-Konventionen	8

Anhang AB32	<u>9</u>
Beispiel AB32-Steuer-File	<u>9</u>
Beispiel AB32-TIC-Modul	<u>10</u>
Beispiel AB32-INI-Routine	<u>11</u>
Beispiel AB32-Task-Manager	<u>12</u>
Beispiel AB32-Unterprogramme	<u>13</u>
Beispiel AB32-Vektor-Block	<u>13</u>
Dummy-Interrupt	<u>14</u>
Anhang Qx	<u>14</u>
Beispiel Qx-Steuer-File	<u>14</u>
Beispiele Qx-TIC-Modul	<u>16</u>
Beispiel Qx-INI-Routine	<u>17</u>
Beispiel Qx-Task-Manager	<u>18</u>
Beispiel Qx-Unterprogramme	<u>19</u>
Beispiel Qx-Vektor-Block	<u>19</u>
Dummy-Interrupt	<u>20</u>

Einleitung

Die Anwendung von Microcontrollern in heutigen Geräten und System nimmt ständig zu. Vielfach sind ähnliche Aufgaben zu bewältigen so dass es sinnvoll ist ein einfaches Betriebs-System und einige Standard-Funktionen anzuwenden.

Das hier vorgestellte Betriebs-System ermöglicht es mit Hilfe eines Task-Managers Kommunikations-Abläufe, periodische Vorgänge und anderes auf einfache Art und Weise zu implementieren. Es wurde bewusst in Assembler geschrieben um eine möglichst schnelle Ausführung der Befehle zu ermöglichen. Zudem bleibt so der ROM-Bedarf in bescheidenem Rahmen.

Die Entwicklungs-Umgebung

Die hier eschriebene Version ist auf die Entwicklungsumgebung von P&E zugeschnitten, wird mit kleinen Anpassungen aber auch auf andern Entwicklungs-Umgebungen lauffähig sein.

Die Files sind als Include-Files konzipiert, so dass im Steuer-File die benötigten Module eingebunden und die Variablen definiert werden. Dies hat den Vorteil, dass alle Module ohne Versions-Verwaltung schnell und einfach assembliert werden.

Das Grundsystem

Das Grundsystem besteht aus dem Steuerfile, TIC-Erzeugung, INI-Block, Task-Manager und Vektor-Definitionen.

- Im Steuerfile werden die Variablen und Konstanten definiert und die benötigten Module eingebunden.
- Die TIC-Erzeugung wird für den jeweiligen Controller-Typ angepasst um das optimale Timer-Interface für diese Aufgabe zu verwenden.
- Die INI-Routinen dienen zur Initialisierung der Variablen und der benötigten Sub-Systeme.
- Der Task-Manager wird mit den nötigen Befehlen ergänzt, um die gewünschten Funktionen auszuführen.
- Im Vektorblock sind die benötigten Interrupt-Vektoren eingetragen.

Der Anwender kann nun seine Befehle und Abläufe in eigenen Modulen programmieren und mit Hilfe des Task-Managers den Programm-Ablauf steuern.

Zusatz-Module

Es stehen verschiedene Zusatzmodule zur Verfügung:

- Mathematik-Modul
- Umwandlungs-Module
 - BIN-BCD
 - BCD-ASCII
 - BCD-7Segment
 - SINGLE (Floatig-Point) in LONG
- Tastatur-Modul für 4x4 Matrix-Tastaturen
- LCD-Ausgabe-Modul (4-bit Mode)

- Uhr-Modul mit Sekunde, Stunde, Monat, Jahr (incl. Schaltjahr) und Wochentag
- Segment-Multiplex-Modul für Anzeigen mit 8-Digits und 8-Segmente
- EEPROM-Programmierungs-Routinen
- Kommunikations-Routinen
 - serielle Kommunikation
 - CRC-Routinen
 - MIP-Bus-Protokoll (RS485) MAXXON-Motors
- und weitere

Beschreibung des Systems

Steuer-File

Das Steuer-File dient einerseits zur Deklaration der Konstanten und Variablen des Programms. Des weiteren werden die benötigten Module und Unterprogramme mittels "include" ins System eingebunden.

[Beispiel QX-Steuer-File](#)

[Beispiel AB32-Steuer-File](#)

TIC-Modul

Das TIC-Modul erzeugt die für den Task-Manager benötigten Flags. Die Grundlage für das TIC-Intervall ist die im Steuermodul definierte Konstante TIC. Das TIC-Intervall ist abhängig von der Controller-Taktfrequenz und dem verwendeten Timer-System.

So beträgt bei der Qx-Familie die Oszillatorfrequenz 12.8MHz (nominal). Die System-Frequenz ist jedoch viermal kleiner (3.2MHz) so dass die Zyklus-Zeit 321ns beträgt. Für ein TIC-Intervall von 1ms wäre demzufolge

TIC EQU 3200

einzutragen. Die Konstanten TICH und TICL werden beim Assemblieren berechnet.

Es werden folgenden Zähler und Flags verwaltet:

Z_CORE	der Core-Timer wird bei jedem TIC-Interrupt inkrementiert und kann im Task-Manager für Zeitberechnungen herangezogen werden. Bit 0 kann als F_TIC2 benutzt werden.
CORE_F	die Steuer-Flags für den Task-Manager
F_TIC1	Das wichtigste Steuer-Flag für den Task-Manager. Ein Wechsel von "0" auf "1" dieses Flags startet die Task-Verarbeitung für den aktuellen Zeitabschnitt.

Die folgenden Flags signalisieren den Start des jeweiligen Zeitabschnitts. Diese Flags sind jeweils für einen TIC aktiv.

aktiv jeden

F_TIC4	4. TIC
F_TIC8	8. TIC
F_TIC16	16. TIC
F_TIC32	32. TIC
F_TIC64	64. TIC
F_TIC128	128. TIC
F_TIC256	256. TIC

Das TIC-Modul muss für den jeweiligen Controller angepasst werden, da nicht alle Controller mit den gleichen Timer-Interfaces ausgerüstet sind.

[Beispiel QX-TIC-Modul](#)

[Beispiel AB32-TIC-Modul](#)

INI-Routine

Die INI-Routine dient zum Initialisieren des Systems. Die verschiedenen Sub-Systeme müssen initialisiert werden. Dazu gehört immer auch die Initialisierung der TIC-Erzeugung, des Watch-Dogs und der Task-Manager-Flags. Für die Initialisierung der Variablen und der andern Sub-Systeme ist der Anwender verantwortlich.

[Beispiel QX-INI-Routine](#)

[Beispiel AB32-INI-Routine](#)

TASK-Manager

Der Task-Manager steuert den Programm-Ablauf des Systems. Dem Anwender stehen die verschiedenen Flags und der Core-Timer zur Steuerung des Ablaufs zur Verfügung. Normalerweise wird mit Hilfe der Task-Manager-Flags (TASK_F1) angezeigt in welchem Zustand das System sich befindet.

Zum Beispiel:

- Warten auf ein Ereignis
- Am Ausführen einer Aktion
- Aktion wird angefordert
- etc

Funktionsweise:

Der Taskmanager wartet in einer Schleife bis das Flag TIC_1 in C_FLAGS von "0" auf "1" wechselt. Nach dem Rückstellen der Watch-Dogs werden die einzelnen Task-Abschnitte abgearbeitet. Am Schluss aller Abschnitte wird erneut in die Warteschleife gesprungen.

In den Task-Abschnitten können die einzelnen Aktionen ausgeführt werden, wobei mit Hilfe der TIC-Flags und anderen Flags entschieden werden kann, was genau ausgeführt werden muss. Es ist möglich, alle Task-Abschnitte sequenziell auszuführen, oder aber nur eine Aktion pro TIC zuzulassen indem einfach nach der Ausführung der Aktion ans Ende des Task-Managers springt. Es ist darauf zu achten, dass die Rechenzeit der einzelnen Aktionen die TIC-Dauer nicht übersteigt. Sollte dies aufgrund verschiedener Interrupts etc. trotzdem vorkommen funktioniert das

System dennoch einwandfrei, da die TIC-Routine weiterarbeitet. Lediglich das Timing des Task-Managers wird beeinflusst.

Bei Bedarf können auch weitere Falgs definiert werden. Die Namen der Speicherstellen enden im EBS08 normalerweise mit_F.. und die Flags selbst beginnen mit F_....
Im Beispiel-Programm sind zwei Flags F_AKT1 und F_Akt2 im Byte TASK_F1 definiert

Die Namen der einzelnen Task-Abschnitte beginnen mit T_.... und enden mit T_.....E.

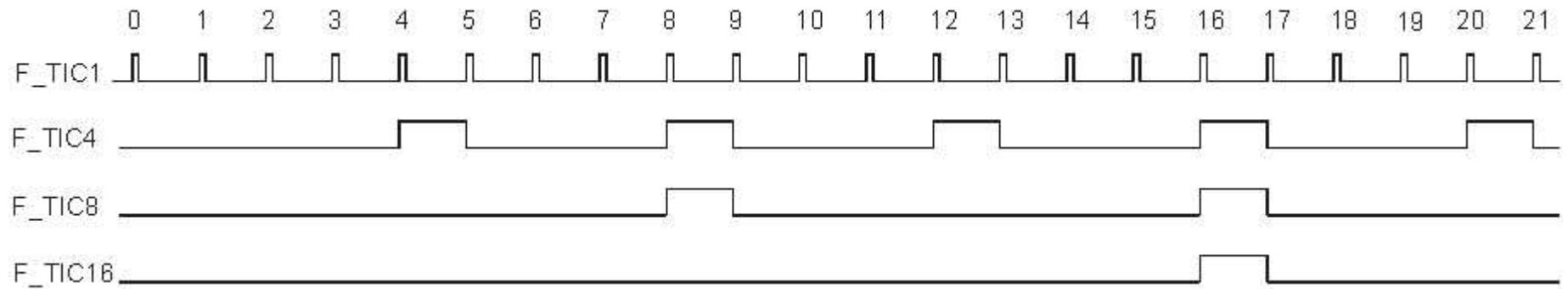
[Beispiel QX-Task-Manager](#) [Beispiel AB32-Task-Manager](#)

Vektor-Block

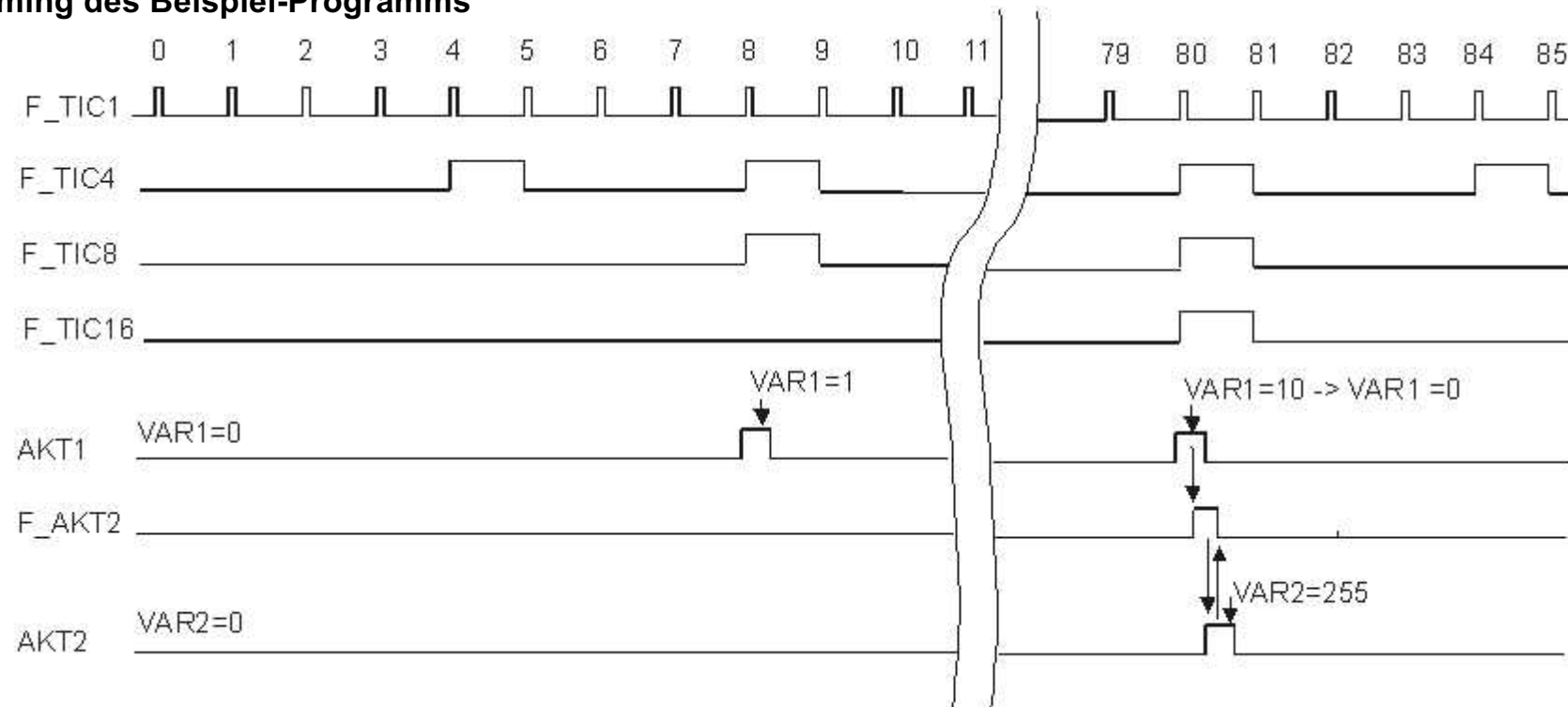
Im Vektor-Block werden die benötigten Interrupt-Vektoren definiert. Für ein einfaches System genügt ein Timer-Vektor und der Reset-Vektor. Werden für die Programm-Ausführung weitere Interrupts benötigt, so sind diese mit dem Namen der Routine einzutragen. Im EBS08-System beginnen die Namen der Interrupt-Routinen jeweils mit "I_". Nicht definierte Interrupts werden in eine "Dummy"-Routine geleitet, die nur aus dem Befehl RTI besteht und sicherstellt, dass das Programm sich nicht "aufhängt".

[Beispiel QX-Vektorblock](#) [Beispiel AB32-Vektor-Block](#)

EBS08-Timing



Timing des Beispiel-Programms



Namens-Konventionen

generell:	Beispiel
Namen von Variablen werden gross geschrieben	VAR1
Unter-Programme beginnen mit dem Modul-Namen	SUB1_AKT1
und werden durch ein Label mit dem selben Namen+"E" beendet	SUB1_AKT1E
wobei SUB1 der Name des Moduls und AKT1 der Name der Prozedur ist.	

Ausnahmen bilden die Interrupt-Routinen und die Task-Manager-Abschnitte

im Speziellen:

Zähler	<i>Z_name</i>	Z_CORE
Flag-Bytes	<i>name_F</i>	CORE_F
Flags	<i>F_name</i>	F_TIC1
Interrupt-Routinen	<i>I_name</i>	I_TIC
Task-Manager-Abschnitt	<i>T_name</i>	T_AKT1

Anhang AB32

Beispiel-Listings für den AB32-Microcontroller

Beispiel AB32-Steuer-File

[zur Beschreibung des Steuer-Files](#)

```
* ab_beisp.asm
* erstellt:      09.04.2004 st-js
* ergänzt:

$BASE 10T      ;Dezimalbasis
$PAGEWIDTH 120

VERS EQU 01          ;Versionsnummer

$NOLIST          ;Register-Files nicht in Listing einbinden

*Register          ;Register und Memory-Definitionen

#include "c:\hc08\def\hc08ab32.sym"
#include "c:\hc08\def\hc08ab32.mem"

$LIST

* Systemtakt 4.9152 MHz
* Prozessor-Zyklus Tp = 813,8ns (1/f * 4)

* 1 ms = 1228 Zyklen

TIC EQU 1228
TICH EQU {TIC\256} ;high = TIC/256
TICL EQU TIC-TICH*256 ;low = TIC-TICH*256

WDOG EQU COPCTL ;WDOG Zuweisung

*Speicherbelegung

ORG RAM

CORE_F RMB 1 ;Core Flags Bit0 = 1 => C_Timer neu (.25ms)
F_TIC1 EQU 0 ;wenn 1 dann neuer TIC (Manager setzt Flag zurück)
;nachfolgende Flags werden von der TIC-Routine verwaltet
;wenn gesetzt dann:
F_TIC4 EQU 1 ;4 TIC's vorbei 1s
F_TIC8 EQU 2 ;8 TIC's vorbei 2 ms
F_TIC16 EQU 3 ;16 TIC's vorbei 4 ms
F_TIC32 EQU 4 ;32 TIC's vorbei 8 ms
F_TIC64 EQU 5 ;64 TIC's vorbei 16 ms
F_TIC128 EQU 6 ;128 TIC's vorbei 32 ms
F_TIC256 EQU 7 ;256 TIC's vorbei 64 ms

Z_CORE RMB 1 ;Core-Zähler in 1TIC Schritten

TASK_F1 RMB 1 ;Task-Manager Flags1 1=

F_AKT2 EQU 1 ;Anforderung Aktion 2

* temporäre Speicher

MAT0 RMB 1 ;Hilfsvariable für Math-Funktionen
MAT1 RMB 1
MAT2 RMB 1
MAT3 RMB 1
MAT4 RMB 1
MAT5 RMB 1
MAT6 RMB 1
MAT7 RMB 1

TMPA RMB 1
TMPB RMB 1
TMPC RMB 1
TMPD RMB 1
```

```

V0      RMB      1      ;Variable 0
V1      RMB      1      ;Variable 1
V2      RMB      1      ;Variable 2
V3      RMB      1      ;Variable 3
V4      RMB      1      ;Variable 4
V5      RMB      1      ;Variable 5
V6      RMB      1      ;Variable 6
V7      RMB      1      ;Variable 7
V8      RMB      1      ;Variable 8

VRES    RMB      1      ;Resultat

* Variable für A- und X-Register in Interrupt- und Sub-Routinen (wergen HC05-Familie)

TMPRA   RMB      1      ;A Speicher
TMPRX   RMB      1      ;X Speicher

POINTER RMB      2      ;Pointer für HX-Zugriffe

VAR1    RMB      1      ;Variable 1
VAR2    RMB      1      ;Variable 2

        ORG      ROM
$include "c:\hc08\projects\ab_beispiel\ab_beisp_ini.asm"          ;Initialisierung
$include "c:\hc08\projects\ab_beispiel\ab_beisp_tsk.asm"          ;Task-Manager

$incl
ude "c:\hc08\projects\ab_beispiel\ab_beisp_sub1.asm"              ;Unterprogramme

$include "c:\hc08\projects\ab_beispiel\ab_beisp_tic.asm"          ;TIC Erzeugung
$include "c:\hc08\projects\allgemein\i_dummy.asm"                 ;Dummy Interrupt

        ORG      VECTOR
$include "c:\hc08\projects\ab_beispiel\ab_beisp_vec.asm"          ;Interrupt-Vektoren

```

[zum Inhaltsverzeichnis](#)

Beispiel AB32-TIC-Modul

* [ab_beisp_tic](#) [zur Beschreibung des TIC-Moduls](#)

```

* Benötigt:      C_TIMER          ;Timer mit 1ms Auflösung
*               C_FLAGS          ;TIC-Flags
*               TMPRA, TMPRB

* Erstellt:      09.04.2004 st-js
* Ergänzt:

```

* PIT-Interrupt

I_TIC:

I_TIC1:

```

*      TIC
*      realisiert mit Programmable Interrupt Timer
*      wenn ein PIT Interrupt auftritt wird diese Routine
*      ausgeführt.

        LDA      PSC              ;TICs fuer Task-Manager
        BCLR    POF,PSC          ;POF löschen

I_TIC12:
        CLR     CORE_F           ;Flags löschen

```

```

        INC     Z_CORE                ;Z_CORE + 1
        BNE     I_TIC13
        BSET    F_TIC256,CORE_F      ;256 TIC's vorbei

I_TIC13:
        BSET    F_TIC1,CORE_F        ;es sind wieder 1 TIC abgelaufen
                                        ;bereit für den naechsten TIC

        LDA     Z_CORE                ;TIC_Zähler laden

I_TIC14:
        AND     #%01111111          ;maskieren -> 0 = 128 TIC's
        BNE     I_TIC1A
        BSET    F_TIC128,CORE_F

I_TIC1A:
        AND     #%00111111          ;maskieren -> 0 = 64 TIC's
        BNE     I_TIC1B
        BSET    F_TIC64,CORE_F

I_TIC1B:
        AND     #%00011111          ;maskieren -> 0 = 32 TIC's
        BNE     I_TIC1C
        BSET    F_TIC32,CORE_F

I_TIC1C:
        AND     #%00001111          ;maskieren -> 0 = 16 TIC's
        BNE     I_TIC1D
        BSET    F_TIC16,CORE_F

I_TIC1D:
        AND     #%00000111          ;maskieren -> 0 = 8 TIC's
        BNE     I_TIC1E
        BSET    F_TIC8,CORE_F

I_TIC1E:
        AND     #%00000011          ;maskieren -> 0 = 4 TIC's
        BNE     I_TICE
        BSET    F_TIC4,CORE_F

I_TICE:
        RTI                            ;Ende

INI_TIC:
                                        ;Initialisieren der TIC-Routine

        CLR     CORE_F                ;löschen der TIC-Variablen
        CLR     Z_CORE

        BSET    PRST,PSC              ;PIT-Counter Reset
        LDA     PSC                    ;laden PIT Status
        AND     #%111111000
        STA     PSC                    ;PPSx auf 0

        LDA     #TICH                  ;MOD-Register laden
        STA     PMODH
        LDA     #TICL
        STA     PMODL

INI_TICE:
        RTS

START_TIC:
                                        ;Startet TIC
        BSET    PIE,PSC                ;freigeben PIT-Interrupt
        BCLR    PSTOP,PSC

START_TICE:
        RTS

```

[zum Inhaltsverzeichnis](#)

Beispiel AB32-INI-Routine

* [ab_beisp_ini.asm](#)

[zur Beschreibung der INI-Routinen](#)

* Benötigt:

* Erstellt: 09.04.2004 st-js
* Ergänzt:

```
INI_S:
    CLRA
    SEI                ;Interrupts AUS !
    LDHX #RAM_E+1
    TXS                ;Stackpointer auf RAM-Ende schieben
    STA WDOG           ;Watchdog zurücksetzen

INI_T:
    JSR INI_TIC        ;Output-Compare-Zähler stoppen
    CLI

INI_PRT:
    LDA #00000010
    STA PORTA          ;nur PA1 auf 1
    LDA #00101010
    STA DDRA           ;Port A Bit 1,3,5 OUTPUT Rest INPUT
;    LDA #00000000
;    STA PTAPUE        ;Pull-Up alle AUS

INI_VAR:
    CLRA
    STA TASK_F1        ;Task-Manager Variable
    STA VAR1           ;Variable 1 und 2 = 0
    STA VAR2

INI_INT:
    JSR START_TIC     ;TIC starten

INI_WART:
                                ;128 ms warten (fakultativ)
    STA WDOG
    BRCLR F_TIC128,CORE_F,INI_WART

INI_E:
    JMP T_S            ;zum Task-Manager

VER:    DB VERS        ;Versions-Nummer im ROM
```

[zum Inhaltsverzeichnis](#)

Beispiel AB32-Task-Manager

* [ab_beisp_tsk.asm](#)

[zur Beschreibung des Task-Managers](#)

* Task-Manager

```
* Benötigt:
*     CORE_F           ;Timer-Flags
*     Z_CORE           ;Timer-Zähler
```

* Erstellt: 03.03.2004 st-js
* Ergänzt:

```

T_S:   BRCLR   F_TIC1,CORE_F,T_S      ;warten auf Start einer lms-Einheit
       BCLR   F_TIC1,CORE_F          ;Flag löschen

       STA    WDOG                    ;Watchdog zurücksetzen

T_SX:

* hier wird entschieden was während diesem TIC getan werden soll

T_AKT1:   BRCLR   F_TIC8,CORE_F,T_AKT1E ;Aktion 1
          JSR    SUB1_AKT1             ;Aktion 1 nur alle 8 TICs ausführen
          ;ausführen Aktion 1
T_AKT1E:

T_AKT2:   BRCLR   F_AKT2,TASK_F1,T_AKT2E ;Aktion 2
          JSR    SUB1_AKT2             ;Aktion 2 nur ausführen, wenn diese
          ;von Aktion 1 angefordert wird
          ;ausführen Aktion 2
T_AKT2E:

T_E:     JMP     T_S                  ;und von vorne

```

[zum Inhaltsverzeichnis](#)

Beispiel AB32-Unterprogramme

* ab_beisp_sub1.asm

* Benötigt: VAR1, VAR2, TASK_F1

* Erstellt: 03.03.2004 st-js

* Ergänzt:

```

SUB1_AKT1:   ;inkrementiert VAR1
             ;ist VAR1 > 10 dann wird F_AKT2
             ;in TASK_F1 und VAR1 auf 0 gesetzt

             LDA    VAR1
             INCA
             STA    VAR1              ;VAR1 = VAR1+1
             CMP    #10
             BLS    SUB1_AKT1E       ;wenn <= 10 dann erledigt

             BSET   F_AKT2,TASK_F1   ;Aktion 2 anfordern
             CLR    VAR1             ;Variable 1 löschen

SUB1_AKT1E:  RTS

SUB1_AKT2:   ;dekrementiert VAR2
             ;und löscht F_AKT2 in TASK_F1

             DEC    VAR2
             BCLR   F_AKT2,TASK_F1

SUB1_AKT2E:  RTS

```

[zum Inhaltsverzeichnis](#)

Beispiel AB32-Vektor-Block

* ab_beisp_vec.asm

[zur Beschreibung des Vektor-Blocks](#)

* Interrupt-Vektoren für AB32

* Erstellt: 09.04.2004 st-js

* Ergänzt:

```

* Vektoren
      ORG          VECTOR          ;Interrupt Tabelle (ab FFD0)

      DW          I_DUMMY          ;ADC Conversion Complete
      DW          I_DUMMY          ;Kexboard Vector
      DW          I_DUMMY          ;SCI Transmit Vector
      DW          I_DUMMY          ;SCI Receive Vector
      DW          I_DUMMY          ;SCI Error Vector
      DW          I_DUMMY          ;Reserved
      DW          I_DUMMY          ;Reserved
      DW          I_DUMMY          ;Timer B Channel 3 Vector
      DW          I_DUMMY          ;Timer B Channel 2 Vector
      DW          I_DUMMY          ;SPI Transmit Vector
      DW          I_DUMMY          ;SPI Receive Vector
      DW          I_DUMMY          ;Timer B Overflow Vector
      DW          I_DUMMY          ;Timer B Channel 1 Vector
      DW          I_DUMMY          ;Timer B Channel 0 Vector
      DW          I_DUMMY          ;Timer A Overflow Vector
      DW          I_DUMMY          ;Timer A Channel 3 Vector
      DW          I_DUMMY          ;Timer A Channel 2 Vector
      DW          I_DUMMY          ;Timer A Channel 1 Vector
      DW          I_DUMMY          ;Timer A Channel 0 Vector
      DW          I_TIC            ;Programmable Interrupt Timer
      DW          I_DUMMY          ;PLL Vector
      DW          I_DUMMY          ;IRQ Vector
      DW          I_DUMMY          ;SWI Vector
      DW          INI_S            ;Reset Vector

```

[zum Inhaltsverzeichnis](#)

Dummy-Interrupt

```

* I_DUMMY.asm

* Dummy-Interrupt-Routine
* Erstellt: 23.05.00 st-js
* Ergnzt:

* fur unbenutzte Interrupts

```

```

I_DUMMY:
      RTI

```

[zum Inhaltsverzeichnis](#)

Anhang Qx

Beispiel-Listings fur Qx-Familie

Beispiel Qx-Steuer-File

[zur Beschreibung des Steuer-Files](#)

```

* qx-beisp.asm

* erstellt:      03.03.2004 st-js
* ergnzt:      06.04.2004 st-js      OSCTRIM aus INTOSCT laden
*

$BASE 10T      ;Dezimalbasis
$PAGEWIDTH 120

VERS      EQU      01          ;Versionsnummer

$NOLIST          ;Register-Files nicht in Listing einbinden

*Register          ;Register und Memory-Definitionen

#include      "c:\hc08\def\hc08qty.sym"
#include      "c:\hc08\def\hc08qty4.mem"

$LIST

```

* OSC-Frequenz nominal 12.8MHz -> Busfrequenz = OSZ/4 = 3.2 MHz

* -> Takt-Zyklus = 312ns

* 1 ms = 3200 Zyklen

```
TIC      EQU      3200
TICH     EQU      {TIC\256}      ;high = TIC/256
TICL     EQU      TIC-TICH*256    ;low = TIC-TICH*256
```

```
WDOG     EQU      COPCTL          ;WDOG Zuweisung
```

*Speicherbelegung

ORG RAM

```
CORE_F   RMB      1      ;Core Flags Bit0 = 1 => C_Timer neu (.25ms)
F_TIC1   EQU      0      ;wenn 1 dann neuer TIC (Manager setzt Flag zurück)
                                ;nachfolgende Flags werden von der TIC-Routine verwaltet
                                ;wenn gesetzt dann:
F_TIC4   EQU      1      ;4 TIC's vorbei 1s
F_TIC8   EQU      2      ;8 TIC's vorbei 2 ms
F_TIC16  EQU      3      ;16 TIC's vorbei 4 ms
F_TIC32  EQU      4      ;32 TIC's vorbei 8 ms
F_TIC64  EQU      5      ;64 TIC's vorbei 16 ms
F_TIC128 EQU      6      ;128 TIC's vorbei 32 ms
F_TIC256 EQU      7      ;256 TIC's vorbei 64 ms

Z_CORE   RMB      1      ;Core-Zähler in 1TIC Schritten

TASK_F1  RMB      1      ;Task-Manager Flags1 1=

F_AKT2   EQU      1      ;Anforderung Aktion 2
```

* temporäre Speicher

```
MAT0     RMB      1      ;Hilfsvariabele für Math-Funktionen
MAT1     RMB      1
MAT2     RMB      1
MAT3     RMB      1
MAT4     RMB      1
MAT5     RMB      1
MAT6     RMB      1
MAT7     RMB      1
```

```
TMPA     RMB      1
TMPB     RMB      1
TMPC     RMB      1
TMPD     RMB      1
```

```
V0       RMB      1      ;Variable 0
V1       RMB      1      ;Variable 1
V2       RMB      1      ;Variable 2
V3       RMB      1      ;Variable 3
V4       RMB      1      ;Variable 4
V5       RMB      1      ;Variable 5
V6       RMB      1      ;Variable 6
V7       RMB      1      ;Variable 7
V8       RMB      1      ;Variable 8
```

```
VRES     RMB      1      ;Resultat
```

* Variable für A- und X-Register in Interrupt- und Sub-Routinen (wergen HC05-Familie)

```
TMPRA    RMB      1      ;A Speicher
TMPRX    RMB      1      ;X Speicher
```

```
POINTER  RMB      2      ;Pointer für HX-Zugriffe
```

```
VAR1     RMB      1      ;Variable 1
VAR2     RMB      1      ;Variable 2
```

```

        ORG     ROM
$include "c:\hc08\projects\qx_beispiel\qx_beisp_ini.asm"           ;Initialisierung
$include "c:\hc08\projects\qx_beispiel\qx_beisp_tsk.asm"         ;Task-Manager

$include "c:\hc08\projects\qx_beispiel\qx_beisp_sub1.asm"        ;Unterprogramme

$include "c:\hc08\projects\qx_beispiel\qx_beisp_tic.asm"         ;TIC Erzeugung
$include "c:\hc08\projects\allgemein\i_dummy.asm"                ;Dummy Interrupt

        ORG     VECTOR
$include "c:\hc08\projects\qx_beispiel\qx_beisp_vec.asm"         ;Interrupt-Vektoren

        ORG     INTOSCT

* die folgende Anweisung kann auskommentiert werden, wenn Sie einen Programmierer mit
* der Möglichkeit zum Messen des System-Takts verwenden und der die Adresse $FFC0
* mit dem gefundenen Wert programmiert.

* Wenn Sie den Simulator verwenden und Sie die folgende Zeile auskommentiert haben,
müssen
* Sie einen Wert in die Speicherstelle $FFC0 schreiben. Ansonsten stoppt der Simulator
mit
* der Meldung "Attempt to use invalid or uninitialized memory"

;TRVAL    DB      $80                                           ;default OSZ Trim

```

[zum Inhaltsverzeichnis](#)

Beispiele Qx-TIC-Modul

* [qx_beisp_tic](#) [zur Beschreibung des TIC-Moduls](#)

```

* Benötigt:      Z_CORE           ;Timer mit 1ms Auflösung
*               F_CORE           ;TIC-Flags

* Erstellt:      03.03.2004 st-js
* Ergänzt:

```

* TIM-Interrupt

I_TIC:

I_TIC1:

```

*      TIC
*      realisiert mit TIM
*      wenn ein TIM TOF-Interrupt auftritt wird diese Routine
*      ausgeführt.

```

```

                                ;TICs fuer Task-Manager
                                ;TOF löschen
        LDA     TSC
        BCLR   TOF,TSC

```

```

I_TIC12:
        CLR    CORE_F           ;Flags löschen
        INC    Z_CORE           ;C_TIMER + 1
        BNE    I_TIC13
        BSET   F_TIC256,CORE_F  ;256 TIC's vorbei

```

```

I_TIC13:
        BSET   F_TIC1,CORE_F    ;es sind wieder 1 TIC abgelaufen
                                ;bereit für den naechsten TIC

```

```

        LDA    Z_CORE           ;TIC_Zähler laden

```

```

I_TIC14:
    AND    #%01111111          ;maskieren -> 0 = 128 TIC's
    BNE    I_TIC1A
    BSET   F_TIC128,CORE_F

I_TIC1A:
    AND    #%00111111          ;maskieren -> 0 = 64 TIC's
    BNE    I_TIC1B
    BSET   F_TIC64,CORE_F

I_TIC1B:
    AND    #%00011111          ;maskieren -> 0 = 32 TIC's
    BNE    I_TIC1C
    BSET   F_TIC32,CORE_F

I_TIC1C:
    AND    #%00001111          ;maskieren -> 0 = 16 TIC's
    BNE    I_TIC1D
    BSET   F_TIC16,CORE_F

I_TIC1D:
    AND    #%00000111          ;maskieren -> 0 = 8 TIC's
    BNE    I_TIC1E
    BSET   F_TIC8,CORE_F

I_TIC1E:
    AND    #%00000011          ;maskieren -> 0 = 4 TIC's
    BNE    I_TICE
    BSET   F_TIC4,CORE_F

I_TICE:
    RTI                          ;Ende

INI_TIC:
                                ;Initialisieren der TIC-Routine

    CLR    CORE_F                ;löschen der TIC-Variablen
    CLR    Z_CORE

    BSET   TRST,TSC              ;TIM-Counter Reset
    LDA    TSC                   ;laden TIM Status
    AND    #%11111000
    STA    TSC                   ;PSx auf 0

    LDA    #TICH                 ;MOD-Register laden
    STA    TMODH
    LDA    #TICL
    STA    TMODL

INI_TICE:
    RTS

START_TIC:
                                ;Startet TIC
    BSET   TOIE,TSC              ;freigeben TIM TOF-Interrupt
    BCLR   TSTOP,TSC

START_TICE:
    RTS

```

[zum Inhaltsverzeichnis](#)

Beispiel Qx-INI-Routine

* [qx_beisp_ini.asm](#)

[zur Beschreibung der INI-Routinen](#)

* Benötigt:

* Erstellt: 03.03.2004 st-js
 * Ergänzt:

```

INI_S:
    CLRA

    SEI                                ;Interrupts AUS !
    LDHX    #RAM_E+1
    TXS                                ;Stackpointer auf RAM-Ende schieben

    STA    WDOG                        ;Watchdog zurücksetzen

    LDA    INTOSCT                    ;Laden des Oszillator-Korrektur-Faktors
    CMP    #$FF                       ;wenn dieser $FF beträgt, so ist es besser
    BEQ    INI_S1                     ;den Reset-Wert ($80) beizubehalten!
    STA    OSCTRIM                    ;wenn der Wert nicht $FF beträgt wird der im
Flash
                                ;gespeicherte Wert übernommen
                                ;Tipp: programmieren Sie den ermittelten Wert in
                                ;INTOSCT ($FFC0). Er variiert von Prozessor zu
Prozessor
                                ;mit dem MONLINK-Programmer geht das sehr bequem,
da
                                ;dieser den CPU-Takt ermitteln kann!
INI_S1:
*INI_TIC:
    JSR    INI_TIC                    ;Output-Compare-Zähler stoppen

    CLI

INI_PRT:
    LDA    #%00000010
    STA    PORTA                      ;nur PA1 auf 1
    LDA    #%00101010
    STA    DDRA                      ;Port A Bit 1,3,5 OUTPUT Rest INPUT
;    LDA    #%00000000
;    STA    PTAPUE                    ;Pull-Up alle AUS

INI_VAR:
    CLRA
    STA    TASK_F1                    ;Task-Manager Variable

    STA    VAR1                      ;Variable 1 und 2 = 0
    STA    VAR2

INI_INT:
    JSR    START_TIC                 ;TIC starten

INI_WART:
                                ;128 ms warten (fakultativ)
    STA    WDOG
    BRCLR  F_TIC128,CORE_F,INI_WART

INI_E:
    JMP    T_S                        ;zum Task-Manager

VER:    DB    VERS                    ;Versions-Nummer im ROM

```

[zum Inhaltsverzeichnis](#)

Beispiel Qx-Task-Manager

* [qx_beisp_tsk.asm](#)

[zur Beschreibung des Task-Managers](#)

* Task-Manager

* Benötigt:

```

*    CORE_F                ;Timer-Flags
*    Z_CORE                ;Timer-Zähler

```

* Erstellt: 03.03.2004 st-js
* Ergänzt:

```
T_S:   BRCLR  F_TIC1,CORE_F,T_S   ;warten auf Start einer lms-Einheit  
       BCLR   F_TIC1,CORE_F     ;Flag löschen
```

```
       STA    WDOG                ;Watchdog zurücksetzen
```

T_SX:

* hier wird entschieden was während diesem TIC getan werden soll

```
T_AKT1:   BRCLR  F_TIC8,CORE_F,T_AKT1E   ;Aktion 1  
          JSR    SUB1_AKT1                ;Aktion 1 nur alle 8 TICs ausführen  
          JSR    SUB1_AKT1                ;ausführen Aktion 1
```

```
T_AKT1E:                                     ;Aktion 2  
T_AKT2:   BRCLR  F_AKT2,TASK_F1,T_AKT2E  ;Aktion 2 nur ausführen, wenn diese  
          JSR    SUB1_AKT2                ;von Aktion 1 angefordert wird  
          JSR    SUB1_AKT2                ;ausführen Aktion 2
```

T_AKT2E:

```
T_E:     JMP     T_S                    ;und von vorne
```

[zum Inhaltsverzeichnis](#)

Beispiel Qx-Unterprogramme

* qx_beisp_sub1.asm

* Benötigt: VAR1, VAR2, TASK_F1

* Erstellt: 03.03.2004 st-js
* Ergänzt:

```
SUB1_AKT1:                                     ;inkrementiert VAR1  
                                               ;ist VAR1 > 10 dann wird F_AKT2  
                                               ;in TASK_F1 und VAR1 auf 0 gesetzt  
       LDA    VAR1  
       INCA  
       STA    VAR1                        ;VAR1 = VAR1+1  
       CMP    #10  
       BLS    SUB1_AKT1E                  ;wenn <= 10 dann erledigt  
       BSET   F_AKT2,TASK_F1              ;Aktion 2 anfordern  
       CLR    VAR1                        ;Variable 1 löschen
```

```
SUB1_AKT1E:                                     ;Aktion 1 erledigt  
       RTS
```

```
SUB1_AKT2:                                     ;dekrementiert VAR2  
                                               ;und löscht F_AKT2 in TASK_F1  
       DEC    VAR2  
       BCLR   F_AKT2,TASK_F1
```

```
SUB1_AKT2E:                                     ;Aktion 2 erledigt  
       RTS
```

Beispiel Qx-Vektor-Block

* qx_beisp_vec.asm

[zur Beschreibung des Vektor-Blocks](#)

* Interrupt-Vektoren für AB32
 * Erstellt: 01.12.2003 st-js
 * Ergänzt:

```
* Vektoren
   ORG      VECTOR                ;Interrupt Tabelle (ab FFD0)

   DW      I_DUMMY                ;ADC Conversion Complete 15
   DW      I_DUMMY                ;Keyboard Vector          14
   DW      I_DUMMY                ;not used                 13
   DW      I_DUMMY                ;not used                 12
   DW      I_DUMMY                ;not used                 11
   DW      I_DUMMY                ;not used                 10
   DW      I_DUMMY                ;not used                 09
   DW      I_DUMMY                ;not used                 08
   DW      I_DUMMY                ;not used                 07
   DW      I_DUMMY                ;not used                 06
   DW      I_TIC                  ;Timer Overflow Vector    05
   DW      I_DUMMY                ;Timer Channel 1 Vector   04
   DW      I_DUMMY                ;Timer Channel 0 Vector   03
   DW      I_DUMMY                ;not used                 02
   DW      I_DUMMY                ;IRQ Vector               01
   DW      I_DUMMY                ;SWI Vector               00
   DW      INI_S                  ;Reset Vector
```

[zum Inhaltsverzeichnis](#)

Dummy-Interrupt

* I_DUMMY.asm
 * Dummy-Interrupt-Routine
 * Erstellt: 23.05.00 st-js
 * Ergänzt:
 * für unbenützte Interrupts

I_DUMMY:
 RTI

[zum Inhaltsverzeichnis](#)