

U5 DSP56002 Assembler

Ziel

Erstellen und Austesten eines Kleinprogrammes in Assembler mit dem EVM DSP56002 Board unter Verwendung der Tasking Entwicklungsumgebung (EDE).

Umfeld

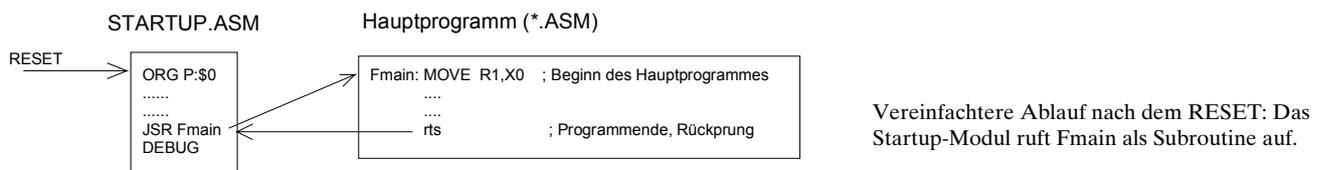
Der Einsatz von Assembler in einem Programmierprojekt kann durchaus sinnvoll sein um zeitkritische Programmteile zu optimieren, d.h. Flaschenhalse wegzuschaffen. Wo immer möglich sollte aber die Codierung in Hochsprache erfolgen.

Beim Test mit dem Debugger wird der Code bis auf Assemblerniveau aufgelöst. Schwierige Fehler können oft nur mit dem tieferen Verständnis und Analyse des Assemblercodes gefunden und behoben werden.

Startup-Code

Normalerweise muss dem eigentlichen Programm ein Startup-Code beigefügt werden der beim RESET des Prozessors gewisse Initialisierungen vornimmt, wie Stack setzen, Speicherbereiche initialisieren, etc. Dieser „Boot-Code“ wird durch das File `STARTUP.ASM` dargestellt. Es beginnt beim RESET-Vektor `P:$0` den Programmablauf, führt die Initialisierungen durch und ruft das eigentlich auszuführende Programm auf, indem das Symbol `Fmain` als Subroutine angesprungen wird.

Das File `STARTUP.ASM` ist daher dem Projekt im Quell- oder Objektcode zuzufügen.



(Das File `STARTUP.ASM` ist bei den Beispielen unter `\asm` zu finden)

Aufgaben

1. Aufsetzen eines neuen, leeren Assembler-Projektes (z.B. `Uebung5_Add2Sum_1`, keine Leerschläge erlaubt. Vgl. hierzu auch Übung 1.
Codieren des ersten Assembler-Beispiels und Austesten mit Crossview.
2. Codieren des zweiten Assembler Beispiels. Achtung: Beim Linken dürfen keine Fehlermeldungen erscheinen!
3. Codieren Sie das Programm zur Filterung in Kap. 6 des EVM Quick Start Guide so, dass es mit der Tasking-EDE erstellt und getestet werden kann. (Zu finden in Motorola HTML-Hilfe, EVM user's guide)

Beispiele

Assemblerprogramme können direkt in der Tasking EDE erstellt werden. Die Assembler Quelltexte erhalten die Extension `.asm` worauf sie automatisch vom `as56` Assembler erkannt und bearbeitet werden. Der Tasking Assembler ist weitgehend kompatibel zum Motorola Assembler, hat aber zahlreiche Erweiterungen. Die Online-Hilfe ist ausgezeichnet, wenn gewisse Grundkenntnisse bestehen.

Für einfache reine Assemblerprogramme sind nur in den Linkeroptionen die C-Libraries vom Linken auszuschalten (vgl. Screenshot in Beispiel 2)

Beispiel 1

Ein Einfachstbeispiel zum Addieren zweier Zahlen welche im Initialisierten X-/Y-Speicher liegen: Da hier weder Stack, noch initialisierte Speicher benötigt werden, kann auf das Startup-Modul verzichtet werden:

```
*****
; Add2Sum.ASM: Demonstrationsprogramm zur Codierung in DSP65002 Assembler
; mit TASKING-AS56 Assembler. Der Programmlauf kann schrittweise
; mit dem Crossview-Debugger verfolgt werden.
; Arbeitsweise: Das Programm addiert zwei Zahlen, welche als direkte Operanden zu
; einer Summe verrechnet werden, die im Y-Speicher abgelegt wird.
; Da hier keine Initialisierten Speicher verwendet werden kann
; ohne Startup-Modul gearbeitet werden. Der Programmbeginn muss
; aber F_START heissen.
;
; <Lauffaehig>
;
; Autor: Gerhard Krucker
; Datum: 18.1.2001
; Assembler: Tasking 2r8
;
; Target: DSP56002 EUM Board, Split Memory
*****
; Speicherdefinitionen
; Y-Datenspeicherbereich <RAM>
;
;      org y:$0          ; Nachfolgende Definitionen fuer Y-Speicher werden ab
;                        ; Adresse $0 gemacht.
resultat    ds 1        ; Ein Wort Speicherplatz fuer das Resultat
;
; Programm welches nach dem Start ausgefuehrt wird
;
;      org    p, ".text": ; Beginn des nutzbaren Programmspeicherbereiche
;
F_START
; Der Bootcode springt zum Label F_START, dem Beginn des eigentlichen Programmes
; Ersten Operand holen
; Zweiten Operand holen
; Addieren
; Summe in resultat speichern
; Hier verbleiben
;
;      jmp *
;
;      end
```

```
2sum_1.asm -fmk6cf8aa.tmp
add2sum_1.out -fmk6cf8ab.tmp
add2sum_1.abs add2sum_1.out -fmk6cf8ac.tmp
: Copy table not referenced, initial data is
0 r8      SN00084745-006 <c> 1997 TASKING, Inc.
```

Hinweis:

Dieses Einfachstbeispiel soll nur Illustrieren, mit wie wenig Aufwand ein Kleinprogramm erstellt werden kann und ist als „Murks“ anzusehen. Man beachte, dass `F_START` die Programmstelle `P:$0` verkörpert und der Code hier im Interruptvektorbereich liegt. Daher ist für alle Programme die etwas aufwändigere Struktur des folgenden Beispiels in jedem Fall vorzuziehen.

Beispiel 2

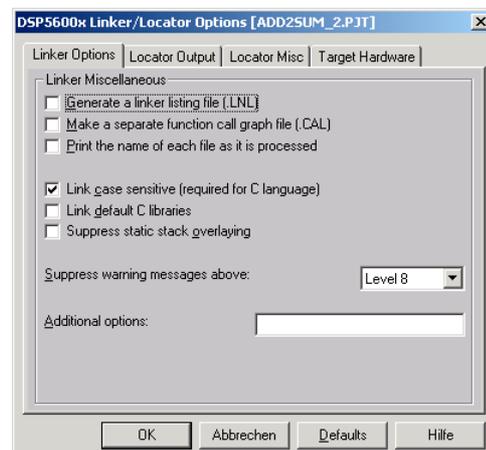
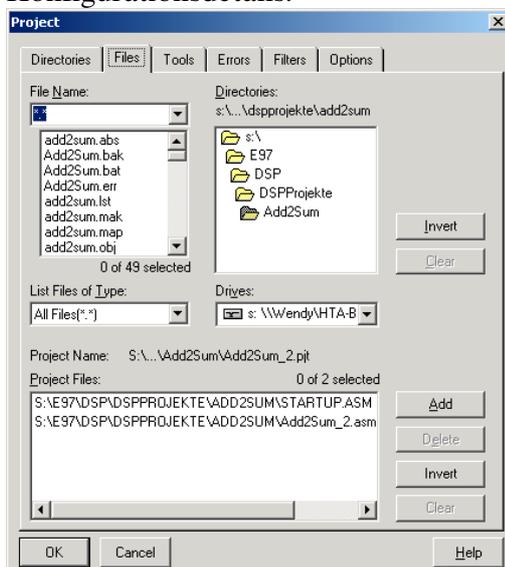
Beispiel mit initialisiertem Speicher und Modul STARTUP.ASM:

```

TASKING EDE [ DSP5600x - S:\E97\DSP\DSPProjekte\Add2Sum\Add2Sum_2.pjt ]
Edit Search Project Text Document Tools Window Help EDE

S:\E97\DSP\DSPPROJEKTE\ADD2SUM\Add2Sum_2.asm
;*****
;
; Add2Sum.ASM: Demonstrationsprogramm zur Codierung in DSP56002 Assembler
; mit TASKING-AS56 Assembler, der programmablauf kann schrittweise
; mit dem Crossview-Debugger verfolgt werden.
;Arbeitsweise: Das Programm addiert zwei Zahlen, welche als Konstanten im X-
; und Y-Speicher gehalten werden zu einer Summe, die wiederum
; in Y-Speicher abgelegt wird.
; Die Konstanten werden durch den Startup-Code beim Programmstart
; in die X- und Y-Speicher kopiert. Weitere Aufgaben des Startupcodes:
; Stack initialisieren, .bss Speicherbereiche loeschen
;
;
; (Lauffaehig)
; Benoetigt: STARTUP.ASM
; Gerhard Krucker
; 18.1.2001
;
; Target: DSP56002 EUM Board, Split Memory
;*****
;
;          global Fmain          ; Programmstart, wird durch den Startupcode als Subroutine aufgerufen.
;                               ; Exportiertes Symbol fuer den Linker.
;
; ; Speicherdefinitionen
; ; Y-Datenspeicherbereich (RAM)
; ; Beispiel zum initialisierten Speicher
;
;          org y, ".ydata":      ; Nachfolgende Definitionen fuer Y-Speicher werden ab
;                               ; im Bereich .ydata abgelegt
;                               ; Das $ bezeichnet einen hexadezimalen Wert.
operand1   dc   $1234
resultat   ds   1              ; Ein Wort Speicherplatz fuer das Resultat
;
; ; X-Datenspeicherbereich
;
;          org x, ".xdata":      ; Nachfolgende Definitionen fuer X-Speicher werden ab
;                               ; im Bereich .xdata abgelegt
operand2   dc   $2345
;
; ; Programm welches nach dem Startupcode ausgefuehrt wird
;
;          org   p, ".ptext":
Fmain      move y:operand1,y0    ; Ersten Operand holen
           move x:operand2,a    ; Zweiten Operand holen
           add  y0,a            ; Addieren
           move a,y:resultat    ; Summe in resultat speichern
           rts                  ; Zurueck zum Startupcode und DEBUG Interrupt ausloesen
;
;
; r:\dsp\c56\bin\as56.exe startup.asm -fmk6dfe6a.tmp
; r:\dsp\c56\bin\as56.exe add2sum_2.asm -fmk6dfe6b.tmp
; r:\dsp\c56\bin\cc56.exe -o add2sum_2.out -fmk6dfe6c.tmp
; r:\dsp\c56\bin\lc56.exe -o add2sum_2.abs add2sum_2.out -fmk6dfe6d.tmp
TASKING program builder v2.0 r8      SN00084745-006 (c) 1997 TASKING, Inc.
    
```

Konfigurationsdetails:



Idee eines Assemblerprogrammes, das zwei Zahlen addiert. Syntaktisch korrekt aber mit diversen konzeptionellen Fehlern:

```
*****
;
; Add2Sum.ASM: Demonstrationsprogramm zur Codierung in DSP65002 Assembler
; mit TASKING-AS56 Assembler, der programmlauf kann schrittweise
; mit dem Crossview-Debugger verfolgt werden.
; Arbeitsweise (Idee): Das Programm addiert zwei Zahlen, welche als Konstante im X-
; und Y-Speicher gehalten werden zu einer Summe, die wiederum
; im Y-Speicher abgelegt wird.
;
; (Nicht Lauffaehig, da verschiedene grobe konzeptionelle Fehler!)
; Gerhard Krucker
; 18.1.2001
; Target: DSP56002 EUM Board, Split Memory
; *****
; Speicherdefinitionen
; Y-Datenspeicherbereich <RAM>
;
;      org y:$0          ; Nachfolgende Definitionen fuer Y-Speicher werden ab
;                        ; Adresse $0 gemacht.
operand1    dc $1234     ; An Adresse y:$0 wird der Wert $1234 abgelegt
;                        ; Das $ bezeichnet einen hexadezimalen Wert.
resultat    ds 1        ; Ein Wort Speicherplatz fuer das Resultat
;
; X-Datenspeicherbereich
;
;      org x:$0          ; Nachfolgende Definitionen fuer X-Speicher werden ab
;                        ; Adresse $0 gemacht.
operand2    dc $2345     ; An Adresse x:$0 wird der Wert $2345 abgelegt
;
; Programm, welches nach dem Start ausgefuehrt wird
;
;      org p:$0          ; RESET Sprungstelle
;      jmp start         ; Interruptvektorbereich ueberspringen
;
;      org p:$40         ; Beginn des nutzbaren Programmspeicherbereiches,
;                        ; Beginn des eigentlichen Programmes
start       move y:operand1,y0 ; Ersten Operand holen
;                        ; Zweiten Operand holen
;      add y0,a          ; Addieren
;      move a,y:resultat ; Summe in resultat speichern
;
;      jmp *             ; Hier verbleiben
;
; end
*****

2sum_bug.asm -fmk620dea.tmp

loop Name
0
0
add2sum.out -fmk620deb.tmp
add2sum.abs add2sum.out -fmk620dec.tmp
Copy table not referenced, initial data is not

Cannot find start label P_START
C:\E97\DSP\DSPPROJEKTE\ADD2SUM\ADD2SUM_BUG.ASM
Build File Find Search Browse Difference Shell
File: S:\E97\DSPPROJEKTE\ADD2SUM\Add2Sum_Bug.asm* Mod Ins Line: 24 Col: 1
```

The screenshot displays the CrossView Pro DSP5600x interface for the file 'Add2Sum_1.abs'. The main window shows assembly code with the following instructions:

```

F_START:      MOVE    #41234,Y0
P:42:        MOVE    #423452,A
P:44:        ADD     YO,A
P:45:        MOVE    A,resultat
P:46:        JMP     $6
P:48:        ** Undefined Opcode **
P:49:        ** Undefined Opcode **
P:4A:        ** Undefined Opcode **
P:4B:        MACR   -Y1,X1,B   X:(R7)+,B   Y:(R3)+,B
P:4C:        MACR   -Y1,X1,B   X:(R7)+,B   Y:(R3)+,B
P:4D:        MACR   -Y1,X1,B   X:(R7)+,B   Y:(R3)+,B
P:4E:        MACR   -Y1,X1,B   X:(R7)+,B   Y:(R3)+,B
P:4F:        MACR   -Y1,X1,B   X:(R7)+,B   Y:(R3)+,B
P:410:       MACR   -Y1,X1,B   X:(R7)+,B   Y:(R3)+,B
P:411:       MACR   -Y1,X1,B   X:(R7)+,B   Y:(R3)+,B
P:412:       MACR   -Y1,X1,B   X:(R7)+,B   Y:(R3)+,B
    
```

The Register window shows the following values:

```

R0 =0003  N0 =ffff  M0 =ffff  R1 =ca66
M1 =ffff  M1 =ffff  R2 =c21c  N2 =ffff
M2 =ffff  R3 =27a6  N3 =ffff  M3 =ffff
R4 =ca66  N4 =ca66  M4 =ffff  R5 =4826
M5 =0814  M5 =ffff  R6 =ffdf  N6 =ffff
M6 =ffff  R7 =f01d  M7 =ffff  M7 =ffff  A2 =00
A1 =024686 A0 =000000 B2 =c8   B1 =46dcc5
B0 =000000 A  =024686000000  PC =0006
B  =46dcc5000000  SP =00   X1 =ca0000
X0 =ca0000  USP=f01d  Y1 =65ca66 Y0 =001234
FP =0000  X  =ca0000ca0000  UFP=f01d
Y  =65ca66001234  CCR=80   LA =ffff
LC =ffff  MR =03   SSL=0003  SSH=0003  OMR=00
SR =03d0  SLeUnzvc
    
```

The Memory window shows a dump of memory addresses from 0x0 to 0x28:

address	+ 0	+ 1	+ 2	+ 3	+ 4	+ 5	+ 6	+ 7
_Y:0x0	0x024686	0x010318	0x8000002	0x200228	0x0021a0	0x140430	0x101222	0x020086
_Y:0x8	0x001000	0x02a102	0x209a02	0x410103	0xe0864e	0x220802	0xc80243	0x088030
_Y:0x10	0x024686	0x012081	0x400439	0x800046	0x0f0001	0x1100a8	0x20012b	0x549120
_Y:0x18	0x104220	0x060406	0x418101	0x4d2000	0xcdc286	0x0010a0	0x101e00	0x180608
_Y:0x20	0x120844	0xe28000	0x0e0441	0x3a0030	0x041040	0x102021	0x180030	0x400a49
_Y:0x28	0x005000	0x080415	0x20b841	0x160250	0x046ecb	0x120e00	0xf00140	0x907915