

16C62 Timer und Interruptfunktionen

Ziel

Verwenden eines Timerinterrupts zur präzisen Zeitverzögerung direkt aus der Hochsprache.

Umfeld

Interrupts werden zur zeitpräzisen Bearbeitung von Ereignissen eingesetzt. Der normale Programmablauf verzweigt bei einer Interruptanforderung zur Interrupt-Serviceroutine, führt diese aus und kehrt nachher zum normalen Programmablauf zurück. Dadurch kann ein definiertes Reaktionszeitverhalten sichergestellt werden.

In vielen Fällen werden Interrupts von Aussen (Periferie) ausgelöst. Dies kann von den Periferiebausteinen selbst oder über eigene Interrupteingänge erfolgen. Weiter sind auch intern erzeugte Software-Interrupts möglich. Sie dienen vorwiegend zur Ausnahmebehandlung, können aber auch direkt vom Programm erzeugt werden.,

Rahmenbedingungen

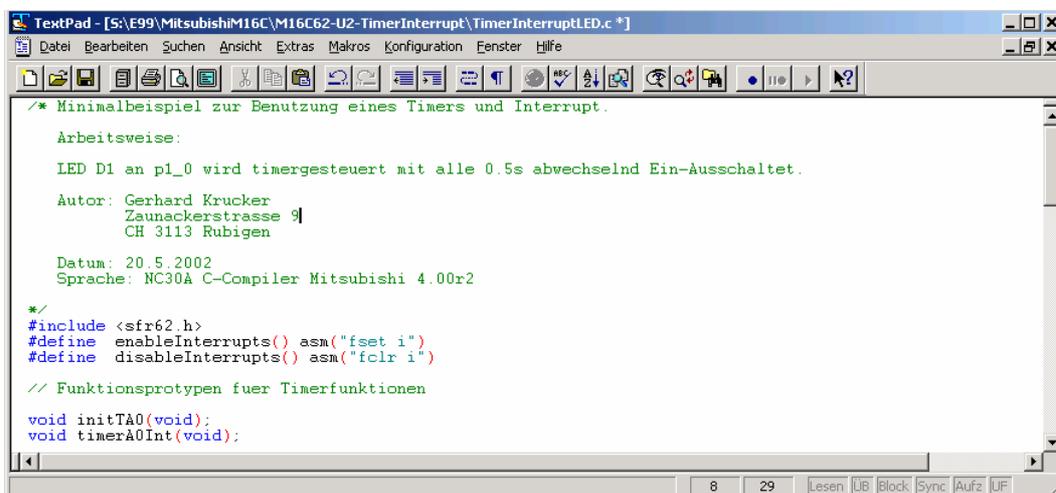
Wird mit Interrupts im eigenen Programm gearbeitet, muss eine Interruptfunktion definiert werden. Sie bearbeitet die Interruptanforderung. Interruptfunktionen sollten generell so kurz und kompakt wie möglich gehalten werden, um das gesamte Zeitverhalten nicht ungünstig zu beeinflussen.

Die Adresse der Interruptfunktion muss in SECT30.INC eingetragen werden.

Die Interruptquellen sind gemäss Vorschrift zu initialisieren. Vor einer Initialisierung immer Interrupts sperren (mit Makro `disableInterrupts`). Nach der Initialisierung Interrupts freigeben.

Beispiel: LED Blinker mit Timersteuerung

1. Neues Projekt nach Uebung 1 erzeugen. (Verwenden Sie kein altes Projekt)
2. Neues C-Sourcefile mit dem Editor erzeugen, benennen und dem Projekt zufügen, z.B. `TimerInterruptLED.C`.
3. Im Sourcefile zuerst die Makros und Funktionsprototypen für die Timerfunktionen definieren:



```
TextPad - [S:\E99\MitsubishiM16C\M16C62-U2-TimerInterrupt\TimerInterruptLED.c *]
Datei Bearbeiten Suchen Ansicht Extras Makros Konfiguration Fenster Hilfe

/* Minimalbeispiel zur Benutzung eines Timers und Interrupt.
Arbeitsweise:
LED D1 an p1_0 wird timergesteuert mit alle 0.5s abwechselnd Ein-Ausschaltet.
Autor: Gerhard Krucker
Zaunackerstrasse 9
CH 3113 Rubigen
Datum: 20.5.2002
Sprache: NC30A C-Compiler Mitsubishi 4.00r2
*/
#include <sfr62.h>
#define enableInterrupts() asm("fset i")
#define disableInterrupts() asm("fclr i")
// Funktionsprototypen fuer Timerfunktionen
void initTA0(void);
void timerA0Int(void);
```

Dann die Timerinitialisierung, Interruptfunktion und Zeitverzögerungsfunktion codieren:

```

TextPad - [S:\E99\MitsubishiM16C\M16C62-U2-TimerInterrupt\TimerInterruptLED.c]
Datei Bearbeiten Suchen Ansicht Extras Makros Konfiguration Fenster Hilfe

// Funktionsprototypen fuer Timerfunktionen
void initTA0(void);
void timerA0Int(void);

volatile unsigned int timer10mSec = 0; // Globale Dekrementvariable in der Timer A0-Interruptroutine

// Initialisierung von Timer TA0 gemaeiss Handbuch "M16C62 data sheet" Seite 82
// Modus: Timer mode autoreload
// Timer Tick bei 10MHz Quarzfrequenz: 10ms

void initTA0(void)
{
    ta0mr = 0x40; // Timer mode, no pulse output, no gate function, cout clk=f8
    ta0=12499; // clk/f8/12500 -> 1/(1+n) n= set value
    udf=0; // Timer count down
    ta0ic = 0x01; // Timer A0 interrupt control: INT level 1
    ta0s = 1; // Timer start flag bit
}

/* Interrupt Service Routine fuer Timer TA0
Beim Underflow, der alle 10ms von Timer TA0 erfolgt, wird der Wert in der globalen Variablen
timer10mSec dekrementiert.
*/
#pragma INTERRUPT timerA0Int // Nachfolgende Funktion timerTA0Int zur Interruptfunktion erklaren
void timerA0Int(void)
{
    if(timer10mSec != 0) {timer10mSec--;}
}

/* void delay(int time)
Zweck: Warten, d.h. Zeitverzoegerung um Parameter time*10ms.
Der Dekrement erfolgt mit Hilfe der globalen Variablen timer10mSec, die in der Interruptfunktion
fuer TA0 alle 10ms dekrementiert wird.
Maximale Verzoegerungszeit: 65535*10ms
*/
void delay(unsigned short t)
{
    timer10mSec=t; // Dekrementvariable setzen
    while (timer10mSec != 0) {}; // Solange noch nicht heruntergezahlt ist, wird gewartet
}
    
```

Die globale Variable `timer10mSec` ist mit dem Prefix `volatile` zu definieren. Damit wird eine Optimierung für diese Variable unterdrückt und der Wert wird bei jedem Zugriff neu aus dem Speicher geladen. Dies ist notwendig, weil ein Schreibzugriff aus dem separaten Interruptprozess erfolgt und die Wertänderung in der Warteschleife bei einer Registervariablen nicht erkannt würde.

Anschliessend das eigentliche Hauptprogramm mit Portinitialisierung auscodieren:

```

TextPad - [S:\E99\MitsubishiM16C\M16C62-U2-TimerInterrupt\TimerInterruptLED.c *]
Datei Bearbeiten Suchen Ansicht Extras Makros Konfiguration Fenster Hilfe

main()
{
    long int i;
    pd1 = 0xFF; // Port 1 alles Ausgaenge
    p1 = 0xFF; // Alle LED D1..D8 aus

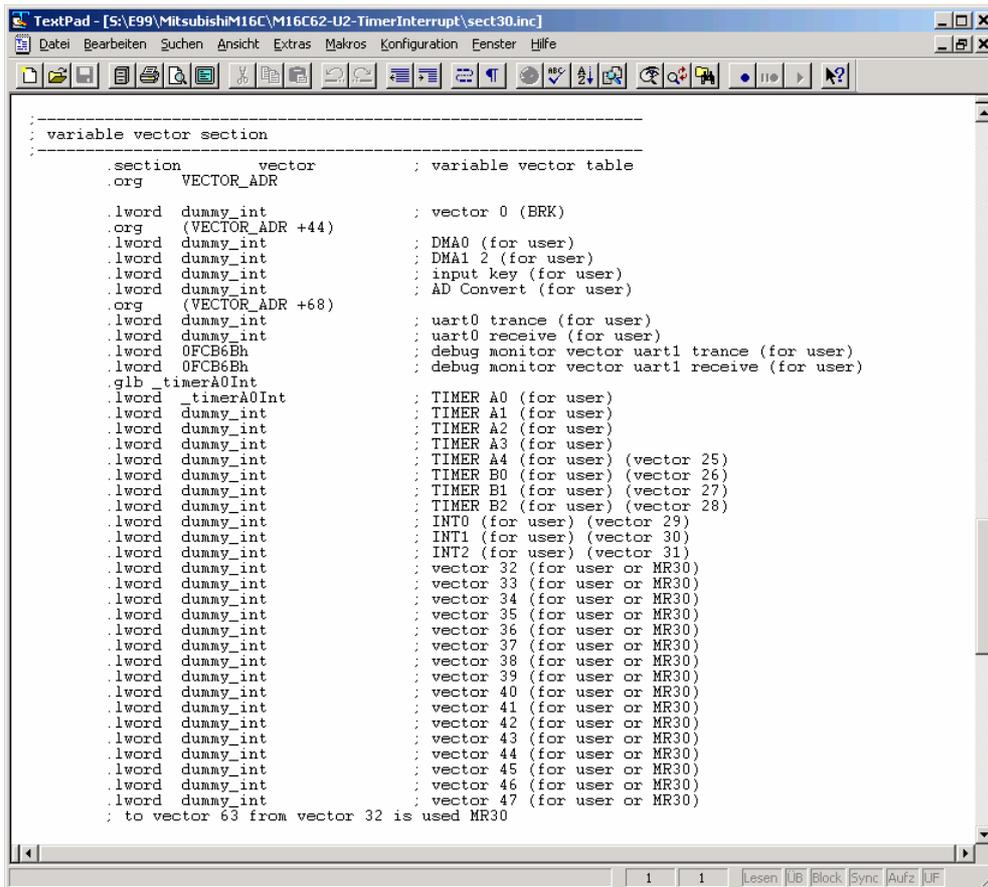
    // Timer TA0 initialisieren und Interrupts freigeben
    disableInterrupts();
    initTA0();
    enableInterrupts();

    for(;;)
    {
        delay(50); // 50*10ms=500ms warten
        p1_0^=1; // LED D1 Status wechseln
    }
}
    
```

- Nun ist der Eintrag der Adresse für die Interruptfunktion für den TA0-Interruptvektor in `SECT30.INC` vorzunehmen. Dazu `SECT30.INC` im Projektmanager öffnen. In der Vektortabelle an der Stelle für `TIMER TA0` den Eintrag `.glb dummy_int` löschen und gegen

```
.glb _timerA0Int
.lword _timerA0Int ; TIMER A0 (for user)
```

ersetzen.



```
-----  
: variable vector section  
-----  
: section      vector      : variable vector table  
: .org        VECTOR_ADR  
  
: .lword      dummy_int     : vector 0 (BRK)  
: .org        (VECTOR_ADR +44)  
: .lword      dummy_int     : DMA0 (for user)  
: .lword      dummy_int     : DMA1 2 (for user)  
: .lword      dummy_int     : input key (for user)  
: .lword      dummy_int     : AD Convert (for user)  
: .org        (VECTOR_ADR +68)  
: .lword      dummy_int     : uart0 trance (for user)  
: .lword      dummy_int     : uart0 receive (for user)  
: .lword      0FCB6Eh       : debug monitor vector uart1 trance (for user)  
: .lword      0FCB6Eh       : debug monitor vector uart1 receive (for user)  
: .glob       _timerA0Int  
: .lword      _timerA0Int    : TIMER A0 (for user)  
: .lword      dummy_int     : TIMER A1 (for user)  
: .lword      dummy_int     : TIMER A2 (for user)  
: .lword      dummy_int     : TIMER A3 (for user)  
: .lword      dummy_int     : TIMER A4 (for user) (vector 25)  
: .lword      dummy_int     : TIMER B0 (for user) (vector 26)  
: .lword      dummy_int     : TIMER B1 (for user) (vector 27)  
: .lword      dummy_int     : TIMER B2 (for user) (vector 28)  
: .lword      dummy_int     : INT0 (for user) (vector 29)  
: .lword      dummy_int     : INT1 (for user) (vector 30)  
: .lword      dummy_int     : INT2 (for user) (vector 31)  
: .lword      dummy_int     : vector 32 (for user or MR30)  
: .lword      dummy_int     : vector 33 (for user or MR30)  
: .lword      dummy_int     : vector 34 (for user or MR30)  
: .lword      dummy_int     : vector 35 (for user or MR30)  
: .lword      dummy_int     : vector 36 (for user or MR30)  
: .lword      dummy_int     : vector 37 (for user or MR30)  
: .lword      dummy_int     : vector 38 (for user or MR30)  
: .lword      dummy_int     : vector 39 (for user or MR30)  
: .lword      dummy_int     : vector 40 (for user or MR30)  
: .lword      dummy_int     : vector 41 (for user or MR30)  
: .lword      dummy_int     : vector 42 (for user or MR30)  
: .lword      dummy_int     : vector 43 (for user or MR30)  
: .lword      dummy_int     : vector 44 (for user or MR30)  
: .lword      dummy_int     : vector 45 (for user or MR30)  
: .lword      dummy_int     : vector 46 (for user or MR30)  
: .lword      dummy_int     : vector 47 (for user or MR30)  
: to vector 63 from vector 32 is used MR30
```

Bemerkungen:

.glob _timerA0Int definiert das C-Symbol timerA0Int für den Linker als extern.

.lword _timerA0Int setzt den Interruptvektor für Timer A0 auf die Adresse Funktion timerA0Int.

Durch den C-Compiler wird allen exportierten Symbolen ein _-Zeichen vorgesetzt.

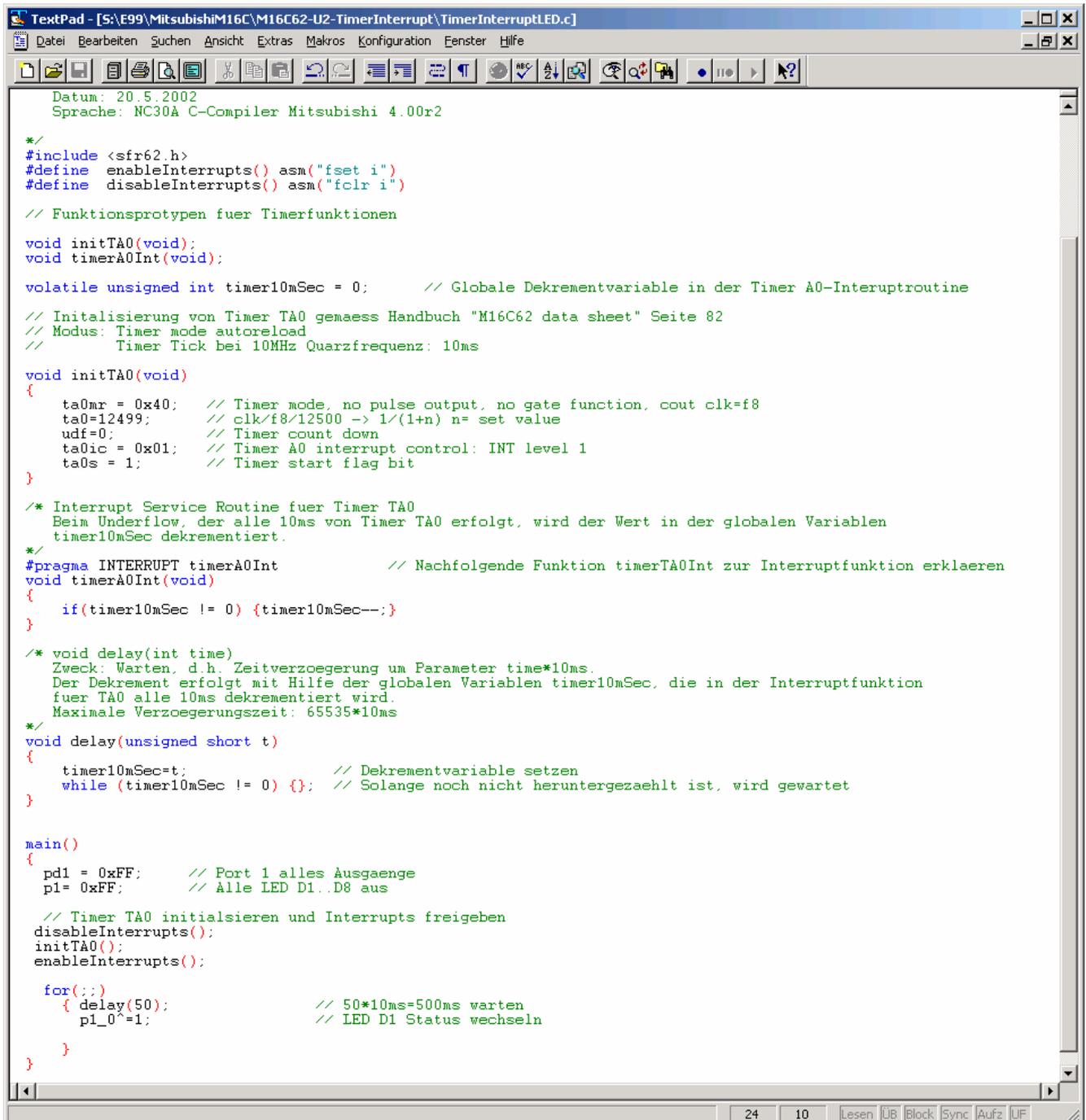
Werden Änderungen in SECT30.INC vorgenommen ist das gesamte Projekt neu zu bauen (Rebuild-Knopf).

5. Programm compilieren, download und testen.

Aufgaben

1. Auscodieren des Beispiels und Test.
2. Änderung des Beispiels so, dass die Blinkfrequenz zwischen 50ms..1s periodisch variiert. Pro Blinkdurchgang wird die Zeitverzögerung um 10ms dekrementiert und beim Erreichen der unteren Grenze inkrementiert.

Gesamtes Beispiel:



```
TextPad - [5:\E99\Mitsubishi\M16C\M16C62-U2-TimerInterrupt\TimerInterruptLED.c]
Datei Bearbeiten Suchen Ansicht Extras Makros Konfiguration Fenster Hilfe

Datum: 20.5.2002
Sprache: NC30A C-Compiler Mitsubishi 4.00r2

*/
#include <sfr62.h>
#define enableInterrupts() asm("fset i")
#define disableInterrupts() asm("fclr i")

// Funktionsprototypen fuer Timerfunktionen
void initTA0(void);
void timerA0Int(void);

volatile unsigned int timer10mSec = 0; // Globale Dekrementvariable in der Timer A0-Interruptroutine

// Initialisierung von Timer TA0 gemaess Handbuch "M16C62 data sheet" Seite 82
// Modus: Timer mode autoreload
// Timer Tick bei 10MHz Quarzfrequenz: 10ms

void initTA0(void)
{
    ta0mr = 0x40; // Timer mode, no pulse output, no gate function, cout clk=f8
    ta0=12499; // clk/f8/12500 -> 1/(1+n) n= set value
    udf=0; // Timer count down
    ta0ic = 0x01; // Timer A0 interrupt control: INT level 1
    ta0s = 1; // Timer start flag bit
}

/* Interrupt Service Routine fuer Timer TA0
Beim Underflow, der alle 10ms von Timer TA0 erfolgt, wird der Wert in der globalen Variablen
timer10mSec dekrementiert.
*/
#pragma INTERRUPT timerA0Int // Nachfolgende Funktion timerTA0Int zur Interruptfunktion erklaren
void timerA0Int(void)
{
    if(timer10mSec != 0) {timer10mSec--;}
}

/* void delay(int time)
Zweck: Warten, d.h. Zeitverzoegerung um Parameter time*10ms.
Der Dekrement erfolgt mit Hilfe der globalen Variablen timer10mSec, die in der Interruptfunktion
fuer TA0 alle 10ms dekrementiert wird.
Maximale Verzoegerungszeit: 65535*10ms
*/
void delay(unsigned short t)
{
    timer10mSec=t; // Dekrementvariable setzen
    while (timer10mSec != 0) {}; // Solange noch nicht heruntergezaehlt ist, wird gewartet
}

main()
{
    pd1 = 0xFF; // Port 1 alles Ausgaenge
    pl= 0xFF; // Alle LED D1..D8 aus

    // Timer TA0 initialisieren und Interrupts freigeben
    disableInterrupts();
    initTA0();
    enableInterrupts();

    for(;;)
    { delay(50); // 50*10ms=500ms warten
      pl_0^=1; // LED D1 Status wechseln
    }
}
```