The **amforth** Cookbook

Author: Erich Wälde
Datum: 2009-02-22 angefangen

Inhaltsverzeichnis

Kapitel 1

Projekt mit ATMEGA 32

1.1 Board

embedit prototype board mit ATMEGA *32, Quarz 11.059200 MHz, serielle Schnittstelle auf 115200 baud*

1.2 Beispiel–Applikation

Erklärung einer kompletten Beispielapplikation; amforth; dictionary; amforth libs; template

1.3 ATMEGA Register– und Bit–Bezeichungen

hier muß ein komplettes amforth file hin, in dem die Register und Bits alle aufgelistet sind.

Kapitel 2

Hardware–lastige Programme

2.1 Hello, world!

Das erste Programm auf einem Mikrocontroller lässt eine LED blinken. Jedenfalls scheint das die Tradition zu sein.

2.2 `turnkey`: wie ein Programm direkt nach dem Einschalten startet

Problem

Das mühevoll geschriebene Programm soll auf dem Mikrocontroller starten, sobald der eingeschaltet wird.

Lösung

Man schreibt eine Funktion, die als `turnkey` registriert wird.
Code Modul/e

```
28 : run-turnkey
29   baud  +usart  >usart  +int
30   ver
31   run
32 ;
33
34 ' run-turnkey is turnkey
```

Diskussion

wort ...

Erklärung, Grenzen, Erweiterungen, Alternativen

Weiterlesen

2.3 Zeit zählen mit *Timer2* und einem Uhrenquarz

Problem

Sie wollen eine verlässliche Messung der Zeit

Lösung

Für eine verlässliche Messung der Zeit kann man *Timer2* mit einem zusätzlichen Uhrenquarz (32768 Hz) betreiben.
Schaltung

- Uhrenquarz an TOSC1,TOSC2

Konfiguration

Um *Timer2* mit dem externen Uhrenquarz zu betreiben, müssen folgende Register/Bits gesetzt werden (Datenblatt S.125ff):

TCCR2: im Timer/Counter Control Register 2 wählt man die ungeteilte Frequenz als Quelle: `CS22,CS21,CS20` = `001`. Die übrigen Bits bleiben auf `0`, d.h. *Timer2* wird im *normal mode* betrieben: das Zählerregister wird von `0` aufwärts bis `255` gezählt, löst dann ggf. einen Interrupt aus und fängt wieder bei `0` an zu zählen.

TCNT2 enthält den aktuellen Zählwert.

ASSR: im Asynchronous Status Register wird das Bit `AS2` auf `1` gesetzt. Damit wählt man den Uhrenquarz an Pin `TOSC1` als Quelle. Das ist der sogenannte `asynchrone Modus`, weil der (externe) Uhrenquarztakt völlig unabhängig vom (internen) Takt des Kontrollers läuft.

TIMSK: im Timer/Counter Interrupt Mask Register wird gesetzt, daß das Überlaufen des Zählers einen Interrupt auslösen soll.

Code

```
1  \ timer2_clock.fs
2
3  \ variable:
4  \     timer2
5  \ words:
6  \     +ticks    register isr and enable interupt
7  \     -ticks    disable interupt
8  \     tick_isr  interupt service routine: increments timer2
9
10 \ f_crystal: 32768 /sec == clock source
```

```
\ overflow:  32768/256 = 128 /sec =^= 7.8125 milli-sec ticks

\ TCCR2 [FOC2,WGM20,COM21,COM20,WGM21,CS22,CS21,CS20]
\ ASSR  [-,-,-,-,AS2,TCN2UB,OCR2UB,TCR2UB]
\ TIMSK [OCIE2,TOIE2,TICIE1,OCIE1A,OCIE1B,TOIE1,OCIE0,TOIE0]

variable timer2

hex
\ Timer2 overflow ISR:
\ increment timer2
: tick_isr
  1 timer2 +!
;
\ enable ticks
: +ticks
  1 TCCR2 c!        \ select clock undivided
  8 ASSR  c!        \ select external quarz clock source
  \ fixme: int!
  ['] tick_isr TIMER2_OVFAddr 2/ int!  \ register ISR
  TIMSK c@ 40 or TIMSK c!  \ enable Timer2 overflow interrupt
;
\ disable ticks
: -ticks
  TIMSK c@
  [ 40 invert ff and ] literal
  and TIMSK c!      \ disable Timer2 overflow interrupt
;
```

Verwendung

```
\ run_timer2_clock.fs

marker --start--

include devices/atmega32.frt
include cooklib/ms.frt           \ ms
include cooklib/timer2_clock.frt \ tick_isr +ticks -ticks
decimal
128 constant ticks/sec
variable old_timer2

: run
  0 timer2 !
  0 old_timer2 !
  +ticks
  decimal
  begin
    1000 ms
```

```
19      cr timer2 @
20        dup u.
21        dup old_timer2 @ -
22          dup .
23        ticks/sec <> if ." <--" then
24      old_timer2 !
25    key? until
26    -ticks
27  ;
```

Ausgabe

```
1   > run
2
3   129 129 <--
4   257 128
5   385 128
6   513 128
7   641 128
8   768 127 <--
9   897 129 <--
10  1025 128
11  1153 128
12  1281 128
13  1409 128
14  1537 128
15  1665 128
16  1793 128
```

Diskussion

Es werden die folgenden Worte definiert:

+ticks konfiguriert *Timer2*. Über das *Timer/Counter Control Register 2* wird festgelegt, daß `Timer2` mit der ungeteilten Frequenz des Uhrenquarzes läuft (`1 TCCR2 c!`). ??? Danach wird `tick_isr` als Interupt Service Routine registriert (`['] tick_isr OVF2addr int!`). Und schließlich wird der *Timer2* Interupt eingeschalten (`TIMSK c@ 40 or TIMSK c!`).

-ticks schaltet den `Timer2` aus.

timer2 ist eine Variable, die beim Ausführen von `tick_isr` hochgezählt wird.

tick_isr ist eine *Interupt Service Routine*, die immer aufgerufen wird, wenn *Timer2* übergelaufen ist und den Interupt auslöst. Die Variable `timer2` wird hochgezählt. Es werden keine Vorkehrungen gegen den Überlauf getroffen.

In der Variablen `timer2` werden die Überläufe von *Timer2* gezählt. *Timer2* wird 32768 pro Sekunde erhöht. Er läuft nach 256 Schritten über, also $32768/256 = 128$ mal pro Sekunde. Dieser Wert wird in der Konstanten `ticks/sec` aufgehoben.

der timer2 interrupt wird nie gelöscht. Daher funktioniert das so. Wenn man ein Bit aus ISR löschen muß, dann muß man die auch in Assembler verfassen. (Beispiel???)

Aysgabebeispiel?

Wenn man die Wartezeit etwas verkürzt, dann kann man das so hinkriegen, daß fast immer 128 ticks vergehen. Man kann aus der Größe der Korrektur auf die Laufzeit des regelmäßigen Programmteils in der Schleife schließen. So ungefähr jedenfalls.

Gibt es Alternativen zu dem o.g. Vorgehen? externe RTC mit Sekunden Interrupt an Pin ???

Weiterlesen

Timer/Counter2 (Datenblatt S.125ff???)
Interrupts und Forth Interupts
Kap. ??? timeup

2.4 Analoge Signalpegel ausmessen (ADC)

Problem

Lösung

Schaltung

Konfiguration

Code Modul/e

Verwendung

Code Verwendung

Diskussion

`wort` ...

Erklärung, Grenzen, Erweiterungen, Alternativen

Weiterlesen

2.5 Puls–Weiten–Modulation (PWM)

Was ist PWM?
Wie geht das beim ATMEGA*?* Die Zähler des ATMEGA können (alle?) auch zum Erzeugen von PWM–Signalen eingesetzt werden. Dabei wird der Zähler mit einem Oszillatorsignal versorgt (normalerweise vom CPU Quarz) und zählt bei jedem Impuls weiter. Es gibt ein oder zwei weitere Vergleichs-Register (output compare register), mit deren Inhalt der Zählerstand bei jedem Schritt verglichen wird. Stimmen die Inhalte von Zähler und Vergleichsregister überein, dann können daraus verschiedene Aktionen automatisch angestoßen werden. Im einfachsten Fall wird der Pegel des zugehörigen Ausgabe-Pins geändert. Interrupts können ebenfalls ausgelöst werden.
es gibt mehrere verschiedene Modi, in denen eine Zähler/PWM Kombination betrieben werden kann.

2.5.1 PWM Signal erzeugen (fast pwm mode)

Problem

Sie wollen ein einstellbares PWM–Signal ausgeben um damit die Helligkeit einer LED (oder die Leistungsaufnahme eines anderen Verbrauchers) zu steuern.
Konkret: Zähler 0 soll ein 8–bit PWM Signal generieren und invertiert auf Pin OC0 ausgeben.

Lösung

Schaltung?
Code

```
: pwm0.init
  0 TCNT0 c!                  \ clear counter
  [ WGM01 bv    WGM00 bv or   \ fast pwm mode
    COM01 bv or COM00 bv or   \ inverted output
    CS01  bv or               \ prescaler 8
  ] literal
  TCCR0 c!
;
hex
variable pwm0
: pwm0!   pwm0 @ OCR0 c! ;

PORTB 3 portpin: pin_pwm0
: pwm0.init.pin
  pin_pwm0 high
  pin_pwm0 pin_output
;
```

Diskussion

Weiterlesen

Kapitel 3

software–lastige Programme

3.1 Uhrzeit/Kalender, `timeup`

Problem

Sie wollen die Zeit in den üblichen Größen Jahr, Monat, Tag, Stunde, Minute, Sekunde verwalten. Die Zeit soll mit timer2 gezählt werden, die Verwaltung der Zeit soll in der Hauptschleife miterledigt werden.

Lösung

timeup

Diskussion

aufwendig, aber gut.

Weiterlesen

- Zeit zählen mit timer2
- i2c
- pcf8583–Uhr am i2c–Bus

3.2 Das DCF Zeitzeichen empfangen und auswerten

3.3 Die `timeup`–Uhr mit der DCF Uhr synchronisieren