AVR-Mikrocontroller in BASCOM programmieren, Teil 2

13. Interrupts

13.1 Externe Interrupts durch Taster

Wenn Taster mittels Polling abgefragt werden, wie in Teil 1 beschrieben, kann das Hauptprogramm entweder nicht viel anderes tun (was auch mal vorkommen kann), oder es besteht die Möglichkeit, dass Tastendrücke verpasst werden wenn der Controller gerade mit etwas anderem beschäftigt ist.

Als Demo für die Arbeit des Controllers soll uns eine blinkende LED dienen.
Das Blinken soll mit 2 Tastern ein und aus geschaltet werden können.

Ohne Interrupts: so geht es nicht.

```
$regfile = "m8def.dat"
$crystal = 8000000
$hwstack=100

'Taster mit Pullup
Config Pind.2 = Input                    'Ein
Config Pind.3 = Input                    'Aus
Portd.2 = 1
Portd.3 = 1

Dim Ledon As Bit


'LED
Config Portd.5 = Output

Do
   If Pind.2 = 0 Then Ledon = 1
   If Pind.3 = 0 Then Ledon = 0

   Portd.5 = Ledon
   Waitms 1000
   Portd.5 = 0
   Waitms 1000
Loop
```

Die 1-bit-Variable Ledon ist ein Flag für Ein/Aus der LED.

Während den Wait-Befehlen kann der Controller nichts nützliches tun , also auch nicht eine betätigte Taste bemerken. Wenn man nicht sehr lange auf die Taste drückt, verpasst das Programm die Betätigung mit großer Wahrscheinlichkeit.

Mit Interrupts
Achtung: es gibt beim Mega8 nur 2 Pins die auf externe Interrupts reagieren können: INT0 = D.2 und INT1 = D.3. Wir können also nicht wie beim Polling irgendwelche Pins nehmen!

Tritt ein Interrupt auf, so unterbricht der Controller das mit dem er gerade beschäftigt war und springt in die Interrupt-Service-Routine. Diese ist ein möglichst kurz gehaltenes Unterprogramm, in dem z.B. ein Flag gesetzt werden kann, welches vom Hauptprogramm ausgewertet wird.

Damit der Controller auf Interrupts reagieren kann, muss man 3 Dinge tun, (hier z.B. für INT0):

- Mit **On Int0** ..... deklarieren wohin beim Interrupt gesprungen werden soll.
- Mit **Enable Int0** sagen dass dieser spezielle Interrupt aktiv ist.
- Mit **Enable Interrupts** dafür sorgen, dass die Interrupts **allgemein** aktiviert sind.

In der Interrupt-Serviceroutine sollte so wenig wie möglich gemacht werden, denn während dieser Zeit ist der Controller für andere Interrupts blockiert.

Der Interrupt kann auf Low-Level (statisch low), Rising (ansteigende Flanke) oder Falling (abfallende Flanke) konfiguriert werden.

Bei Tastern gegen Masse verwendet man „Falling“.

```
$regfile = "m8def.dat"
$crystal = 8000000
$hwstack=100

'Taster mit internem Pullup und aktivierten Interrupts
Config Pind.2 = Input
Config Pind.3 = Input
Portd.2 = 1
Portd.3 = 1
On Int0 Int0service
On Int1 Int1service
Config Int0 = Falling
Config Int1 = Falling
Enable Int0
Enable Int1
Enable Interrupts


'LED
Config Portd.5 = Output

Dim Ledon As Bit

'-----------------------------------------
Do
   Portd.5 = Ledon
   Waitms 1000
   Portd.5 = 0
   Waitms 1000
Loop
'-----------------------------------------
Int0service:
  Ledon = 1
Return

Int1service:
  Ledon = 0
Return
```

Wegen den langen Wait-Befehlen passiert nicht immer sofort etwas, aber der Controller reagiert auch auf einen kurzen Tastendruck.

Noch ein (einfacheres) Beispiel

Das obige Beispiel sollte das Problem verdeutlichen (dass der Controller bei der Polling-Methode nicht immer ansprechbar ist) und eine Lösung zeigen.

Um das Prinzip noch einmal klarzustellen hier ein noch einfacheres Beispiel: die LED soll mit den Tastern nur ein- und ausgeschaltet werden.
In diesem Fall braucht das Hauptprogramm gar nichts zu tun.

```
$regfile = "m8def.dat".dat "
$crystal = 4000000


'Taster
Config Pind.2 = Input
Config Pind.3 = Input
```

```
Portd.2 = 1
Portd.3 = 1
On Int0 Int0serv
On Int1 Int1serv

Config Int0 = Rising
Config Int1 = Falling
Enable Int0
Enable Int1
Enable Interrupts

'LED
Config Pind.5 = Output

'--------------------
Do
  'Hauptprogramm : tu nix oder irgendwas sinnvolles
Loop
'--------------------

Int0serv:
  Portd.5 = 1
Return

Int1serv:
  Portd.5 = 0
Return
```

13.2 Worauf man allgemein bei Interrupts achten muss

- **Der Sprung in die Interrupt-Routine kostet Zeit!**
  BASCOM sichert automatisch immer die Register R0...R31 (außer R6 bis R9 und R12 bis R15) und das Statusregister SREG. Hierfür ist je 1 PUSH-Befehl am Anfang und 1 POP-Befehl am Ende der Serviceroutine erfordert. Für das Retten von SREG wird ein IN und ein PUSH, am Ende ein POP und ein OUT gebraucht.

  Vor dem Eintreten in die Serviceroutine verstreicht mindestens eine Zeit von 50 (für 25 POPs) + 2 (für IN) = 52 Taktzyklen, also z.B. 52 · 0.25µs = 13µs bei einem 4MHz-Quarz!

  Genaueres siehe „Assembler für BASIC-Programmierer“
  http://staff.ltam.lu/feljc/electronics/bascom/ASM_BAS.pdf

  Wenn man den Interrupt mit der Option **Nosave** aufruft, kann man Zeit sparen indem man sich selber um das Sichern der benutzten Register und des Statusregisters kümmert.

- **Das Sichern der Rücksprung-Adresse und der Register braucht Platz in einem Teil des RAM, dem sogenannten Hardware Stack**. Die Größe dieses Speicherbereichs wird in „Options“- „Compiler“ - „Chip“ eingestellt. Es kann sein, dass die Voreinstellung zu klein ist. In dem Fall überlappen HW Stack und BASCOM-Variablen, und es können seltsame Effekte auftreten.

  Empfehlenswert ist es, die Zeile

  ```
  $hwstack = 100
  ```

  an den Programmbeginn zu setzen, so dass auf jeden Fall genug Stack reserviert wird.

13.3 Serieller Interrupt zum Empfangen von Kommandozeichen

Dies ist eine ähnliche Anwendung wie die Steuerung durch Taster.

Vom PC soll ein Kommandobyte gesendet werden, welches im Demobeispiel einfach eine LED schaltet, stellvertretend für eine sinnvolle Arbeit des Controllers.

Die im Teil 1 besprochene Lösung mit Polling wieder den Nachteil, dass Bytes verloren gehen können, wenn der Controller mit etwas anderem beschäftigt ist.

Hier eine Lösung mit Interrupt:

```
'LED über RS232 ein- und ausschalten mit "1" und "0", 'interruptgesteuert
$regfile = "m8def.dat"
$crystal = 4000000
$baud = 9600

On Urxc Serialinterrupt
Enable Urxc
Enable Interrupts

Config Portd.5 = Output                         'LED

Dim Ok As Byte

'-----------------------------------------------------
Do
    'Wenn ein Zeichen da ist
    If Ok <> "" Then
       Select Case Ok
          Case "1"
             Portd.5 = 1
          Case "0"
             Portd.5 = 0
       End Select
    End If
Loop
'-----------------------------------------------------
Serialinterrupt:
   Ok = Udr
Return
```

Hier wurde wieder die Interrupt-Serviceroutine kurz gehalten, und die Auswertung im Hauptprogramm durchgeführt.

13.4 Serieller Interrupt zur interaktiven Kommunikation

Das folgernde Programm reagiert einerseits auf Kommandozeichen und fragt bei bestimmten Zeichen interaktiv nach einer bestimmeten Eingabe.

Im Beispiel wird eine Variable hochgezählt, deren Startwert man resetten oder interaktiv eingeben kann. Um die Übersichtlichkeit zu steigern empfiehlt es sich, von Anfang an mit Unterprogrammen zu arbeiten. Diese können dann später beliebig erweitert werden.

```
$regfile = "m8def.dat"
$crystal = 8000000
$hwstack = 50

$baud = 19200

On Urxc Serialinterrupt
Enable Urxc
Enable Interrupts


Dim I As Byte
```

```
Dim Ok As Byte


Do
   If Ok <> "" Then Gosub Communication
   Waitms 500
   Print I
    I = I + 1
Loop

'-----------------------------------------------------------
Serialinterrupt:
    Ok = Udr
Return
'-----------------------------------------------------------
Communication:
    Select Case Ok
        Case "0"
             I = 0
        Case "i"
            Disable Urxc
            Input "Startwert von I eingeben: " , I
            Enable Urxc
    End Select
    Ok = ""
Return
```

Wichtig: Beim Benutzen des INPUT-Befehls muß der serielle Interrupt abgeschaltet werden!

Ausserdem muß daran gedacht werden, dass die Variable zum Empfang der Zeichen nach der Auswertung rückgesetzt wird.
