Pin Change Interrupt mit BASCOM

DL6GL, 26.05.2023

In den Anfangszeiten, als es neben ATtinys noch die ATmeg8/16/32 gab, musste man sich mit wenigen interruptfähigen Pins zufriedengeben. Beim Atmega8 waren das die INT0- und INT1-Pins PD.2 und PD.3, an denen externe Interrupts ausgelöst werden konnten. Beim ATmega16/32 kam noch INT2 an PB.2 hinzu.

In der Real Time Clock habe ich beim Schaltungsentwurf versucht, am TQFP-Gehäuse benachbarte Pins einzelnen Funktionsgruppen, den Tastern und der RTC, zuzuordnen, um eine spätere Leitungsführung auf dem Board zu erleichtern. Es waren zwei externe Interrupt-Quellen vorzusehen: für den Set-Taster und den RTC-INT-Ausgang. Mit dem INT0/PD2-Pin für den Set-Taster klappte das noch. Neben den Hardware-I2C-Pins PC4 und PC5 zur RTC liegen aber nur normale I/O-Pins. Zeit, sich einmal mit den Pin Change Interrupts zu beschäftigen.

Mit der neueren Generation ATmega 48/88/168/328, die mit dem in der Arduino-Familie verbauten ATmega328P große Verbreitung gefunden hat, gibt es die freie Auswahl aus beliebigen Pins mit dem Pin Change Interrupt. Die Datenblätter geben im Kapitel "External Interrupts" dazu Auskunft. Anders als bei den INT0 bis INT2-Interrupts mit wählbarer Reaktion (ansteigende / abfallende Signalflanke, Low Level oder Change) wird ein Pin Change Interrupt bei jeder Änderung am betreffenden Port ausgelöst.

Eine Konfigurierung hört sich kompliziert an, ist es aber nicht wirklich. Entwicklungsumgebung wie gewohnt: BASCOM-AVR. Aber – das ist in BASCOM wohl nicht zu Ende durchentwickelt. Mit dem bekannten "Config INTx …"wie bei den o.g. INT0 bis INT2 geht es schon mal nicht.

Wir beschränken uns entsprechend der weiten Verbreitung auf die Controller-Familie ATmega 48/88/168/328. Für andere Controller, die Pin Change Interrupts beherrschen, gilt das entsprechend.

Atmel/Microchip hat bei den ATmega 48/88/168/328 mit den Port-Gruppen PB, PC, PD drei Pin Change Interrupt-Gruppen vorgesehen, die von bis zu jeweils 8 I/O-Pins ausgelöst werden können,

Gruppe	Pins	PCINT-Anschlüsse	Bemerkung
PCINT0	PB.0 … PB.7	PCINT0 …PCINT7	
ICINT1	PC.0 … PC.6	PCINT8 …PCINT14	PC.7/PCINT15 fehlt
ICINT2	PD.0 … PD.7	PCINT16…PCINT23	

Damit stehen also, zumindest theoretisch, bis zu 24 Ports zur Verfügung, die einen Interrupt auslösen können.

Im BASCOM-Sprech ginge das dann wie von INT0 bis INT2 gewohnt so, aber ohne "Config INTx":

```
On PCINT1 PCINT1_isr            'Jmp to interrupt service routine
Enable PCINT1
Enable Interrupts
```

Damit ist aber noch nicht festgelegt, welcher der zur PCINT1-Gruppe gehörenden Pins nun einen Interrupt auslösen soll. Hierzu müssen wir das **Pin Change Mask Register 1 (PCMSK1)** noch bemühen, das im Fall PCINT1 den richtigen PCINT-Anschluss dem gewünschten Pin PC.0 bis PC.6 zuordnet. Soll das z.B. der Pin PC.3 sein, müssen wir noch ergänzen:

```
PCMSK1 = &B0000_1000            'Set PCINT11 @ PC.3
```

Für die Gruppen PNINT0 und PCINT2 gibt es entsprechende Register PCMSK0 und PCMSK2.

Wenn uns BASCOM doch noch das explizite Setzen des PCMSKx-Registers abnötigt (habe zumindest nichts dazu gefunden), nehmen wir uns doch gleich alle beteiligten Register vor. Dazu steht erst einmal der Blick ins Datenblatt für ATmega48/88/168/328, Kapitel12.2 an. Komplizierter wird das aber auch nicht.

1 PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0
	-	-	-	-	-	PCIE2	PCIE1	PCIE0
R/W	R	R	R	R	R	R/W	R/W	R/W
Initial	0	0	0	0	0	0	0	0

Uns interessiert der Interrupt PCINT1. Für PCINT2 und PCINT0 gilt entsprechendes.

Hier also **Bit 1 – PCIE1: Pin Change Interrupt Enable 1**

Wenn PCIE1 = 1 gesetzt wird und (!) das Bit I (#7) im Status Register SREG = 1 gesetzt werden, ist der Pin Change Interrupt PCINT1 aktiviert (enabled). Jede Änderung irgendeines PCINT14 … PCINT8-Pins (Ports PC.6 …PC.0) kann damit einen Interrupt erzeugen. Wir setzen also das passende Bit

```
PCICR.PCIE1 = 1                 'Enable PCINT1
```

Wahlweise das ganze Register

```
PCICR = &B0000_0010             'Enable PCINT1
```

Nun müssen wir noch sicherstellen, dass ein solcher Interrupt auch einem Interrupt Request erzeugt, der einen Sprung zu einem Interruptvektor, damit zu einer zugehörigen Interrupt Serviceroutine gewährleistet.

2 PCIFR – Pin Change Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0
	-	-	-	-	-	PCIF2	PCIF1	PCIF0
R/W	R	R	R	R	R	R/W	R/W	R/W
Initial	0	0	0	0	0	0	0	0

Ganz analog setzen wir

```
PCIFR.PCIF1 = 1                 'Enable Interrupt request for PCINT1
```
oder
```
PCIFR= &B0000_0010              'Enable Interrupt request for PCINT1
```

3 I/O-Pin festlegen mit PCMSKx, hier PCMSK1

Pin Change Mask Register 1 – PCMSK1.

Bit	7	6	5	4	3	2	1	0
	-	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8
R/W	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial	0	0	0	0	0	0	0	0

Innerhalb PCINT1 ist noch ein einzelner I/O-Pin auszuwählen:

```
PCMSK1.PCINT11 = 1              'Enable Pin change PCINT11 (PC.3)
```

Wahlweise

```
PCMSK1 = &B0000_1000            'Enable Pin change PCINT11 (PC.3)
```

4 Sprungadresse Interrupt Service Routine

Legen wir noch fest, wohin bei einem Interrupt gesprungen werden soll:

```
On pcint1 pcint11_isr           'Execute PCINT11 interrupt
```

5 Interrupts global freigeben

Der "normale" BASCOM-Programmierer wird das wohl so schreiben:

```
Enable Interrupts               'Enable all Interrupts
```

Das geht auch kürzer mit dem **AVR Status Register SREG**. Uns interessiert hier nur das Bit #7.

```
SREG.7 = 1                      'Enable all Interrupts
```

Wahlweise (SREG Bit 7 heißt "I")

```
SREG.I = 1                      'Enable all Interrupts
```

Damit werden **alle** definierten Interrupts, etwa wie hier von PCINT1, global freigegeben. Realisiert wird das mit UND-Gattern, am Beispiel PCINT1: Der Ausgang des Gatters ist nur dann logisch 1 und löst damit einen Interrupt aus, wenn die Eingänge PCINT1 UND SREG.I beide logisch 1 sind.

In dem Augenblick, wenn eine der definierten Interrupt-Serviceroutinen wie hier "pcint11_isr" angesprungen wird, werden alle anderen Interrupts gesperrt. Das verhindert ein gleichzeitiges Bearbeiten ggf. weiterer anstehender Interrupts: Immer der Reihe nach. Mit Verlassen der aktuellen Interrupt-Serviceroutine über das Return erfolgt wieder die globale Freigabe der Interrupts.
Das Sperren der Interrupts entspricht einem temporären Setzen von SREG.I = 0 während der Ausführung der Interrupt-Serviceroutine.

Ein temporäres Sperren aller Interrupts kann auch im Programm vorgenommen werden, um z.B. kritische Operationen wie z.B. das Beschreiben des EEPROMs oder Ausgaben aktueller Daten auf LCD abzusichern.

Im BASCOM-Sprech:

```
Disable Interrupts              'Disable all Interrupts
```

Oder wie oben:

```
SREG.I = 0                      'Disable all Interrupts
SREG.7 = 0                      'Disable all Interrupts
```

Das ist die Standardvorbesetzung beim Starten des Programms. Eine explizite globale Freigabe von Interrupts ist also erforderlich, sofern Interrupts zu verarbeiten sind.

Nach Abarbeiten des zu schützenden Programmteils müssen anschließend die Interrupts global wieder wie oben freigegeben werden.

Eigentlich wider Erwarten war dann doch festzustellen, dass BASCOM-AVR 2.0.8.3 den Code des Real Time Clock Programms RTC_100_M328.bas mit "Enable / Disable Interrupts" effektiver kompiliert als mit den Registerzuweisungen SREG.7=1 / 0:

- Enable / Disable Interrupts: 9.104 Bytes,
- SREG.7 = 1 / 0: 9.176 Bytes.

Im ersten Fall wurden alle "SREG.7 = 1 / 0" durch "Enable / Disable Interrupts" ersetzt.

Das ganze zusammengerührt:

```
RTC_INT Alias PINC.3             'Input from PCF8563 INT pin
                                 'PC.3 = Input + Pullup
Dim bitAlarm As Bit              '=1: Alarm ON

'Config pin change interrupt at PC.3 (PCTNT11)
PCICR.PCIE1 = 1                  'Enable PCINT1
PCIFR.PCIF1 = 1                  'Enable Interrupt request for PCINT1
PCMSK1.PCINT11 = 1               'Enable Pin change PCINT11
On pcint1 pcint11_isr            'Execute PCINT11 interrupt
SREG.7 = 1                       'Enable all Interrupts

'====================================================================
pcint11_isr:
'Handle PCINT11 interrupt (Port PC.3) triggerd by PCF8563 INT Pin

   If RTC_INT = 0 Then           'PC.3 is low
      bitAlarm = 1               'Alarm ON
   End If
Return
```

In der Do …Loop kann dann auf das Signal bitAlarm = 1 angemessen reagiert werden. Vor der Do …Loop und nach der Alarmbearbeitung ist bitAlarm mit "0" zu initialisieren bzw. zurück zu setzen, etwa so:

```
bitAlarm = 0                     'Initialize Alarm flag = OFF


Do

   'Do something useful here

   If bitAlarm = 1 Then          'External alarm was triggered
      bitAlarm = 0               'Reset Alarm flag

      'Handle alarm, e.g. activate relay, LED, buzzer...

   End If

   'Do some other useful things here

Loop
```