Handhabung von AVR-Registern mit BASCOM

DL6GL, 12.05.2023

BASCOM verbirgt Manipulationen einzelner AVR-Register hinter Hochsprachenbefehlen wie "Config …" oder "Enable …". Mit der Definition des Controllers sind das dann auch gleich die dazu passenden Register und Bits im Register. Die BASCOM-Help bietet eine reiche Auswahl.

Wer genau wissen will, welche Aktionen er womit auslösen wird, muss auf die "atomare" Ebene von Registern und einzelnen Bits heruntersteigen. Die BASCOM-Help hilft da nicht weiter. Das AVR-Datenblatt muss zu Rate gezogen werden. Das ist zu Anfang mühsamer, macht aber den Code transparenter und eröffnet alle Möglichkeiten, die die AVR-Entwickler vorgesehen haben, die aber ggf. von BASCOM aus nicht erreichbar sind.

Die AVR-Register sind i.d.R. 8 Bits breit. Nehmen wir z.B. das im Papier "Pin Change Interrupt" behandelte Register **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

Bit 7: Reserviert, wird immer mit 0 gelesen, keine Schreibänderung.

Jedes der PCINT14 bis PCINT8 Bits wählt einen zugehörigen I/O-Pin mit einer "1" für einen Pin Change Interrupt aus (um die Auswahl wirksam zu machen, muss im Register PCICR – Pin Change Interrupt Control Register das zugehörige Interrupt, hier PCIE1, mit "1" aktiviert werden, in "Pin Change Interrupt" behandelt).

Bei den ATmega 48/88/168/328 stehen für PCINT14 bis PCINT8 die Ports PC.6 bis PC.0. Den Port PC.7 gibt es hier nicht. Um z.B. Port PC.3 für einen Pin Change Interrupt vorzusehen, kann man schreiben (das reservierte Bit 7 ist hier immer "0"):

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

Damit wird das gesamte Register PCMSK1 gesetzt. "&B0000_1000" ist eine leserlichere Schreibweise von "&B00001000". Statt der binären Schreibweise als Spiegelbild der Registerbits könnte man hier auch schreiben (in hex, binär "00001000" = 0x08 hex):

```
PCMSK1 = &H8                        'Set PCINT11 @ PC.3
```
oder auch nur (in dezimal)
```
PCMSK1 = 8                          'Set PCINT11 @ PC.3
```

wie es manche obercoole C-Programmierer belieben zu tun.

Ein *einzelnes Bit* lässt sich in BASCOM auch setzen:

```
PCMSK1.PCINT11 = 1                  'Set PCINT11 @ PC.3
```

Oder auch, PCINT11 ist das Bit Nummer 3 in PCMSK1:

```
PCMSK1.3 = 1                        'Set PCINT11 @ PC.3
```

Hier ist es noch einfach, das gesamte Register zu setzen, da hier gleichartige Bits PCINTxx zu konfigurieren sind. In den meisten Registern gibt es aber Bits mit unterschiedlichen Funktionen, von denen einzelne Bits bei einer gezielten Manipulation unverändert bleiben müssen. Das Beschreiben des gesamten Registers mit einem Schlag scheidet dann also aus. Hier kommen logische Operationen "AND" bzw. "OR" mit Bitmasken zum Zuge.

Zur Erinnerung:

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

X **AND** Y = 1, wenn X **und** Y = 1, sonst = 0

Der **AND**-Operator bietet sich zum **Reset** eines Bits an.

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

X **Or** Y = 1, wenn X oder Y oder beide = 1, sonst = 0

Der **OR**-Operator bietet sich zum **Set** eines Bits an.

Mit einer passenden Bit-Maske und einer AND- oder OR-Operation lassen sich also gezielt einzelne Bits eines Registers bzw. Bytes manipulieren, d.h. auf
1 (= Set) oder 0 (= Reset) setzen am Beispiel MCUCR, ist das MCU Control Register:

```
MCUCR = MCUCR OR &B0110_0000          'Set Bit5 & Bit6 in MCUCR
```

Nur die mit "1" in der Maske besetzten Bits, hier #5 und 6, werden in MCUCR auf "1" gesetzt. Die mit "0" in der Maske besetzten Bits ändern die entsprechenden Bits in MCUCR nicht.

```
MCUCR = MCUCR AND &B1101_1111         'Reset Bit5 in MCUCR
```

Nur die mit "0" in der Maske besetzten Bits, hier #5, werden in MCUCR auf "0" gesetzt.
Die mit "1" in der Maske besetzten Bits ändern die entsprechenden Bits in MCUCR nicht.

Das Set / Reset von ausgewählten Bits mit "OR" oder "AND" erscheint dann vorteilhaft, wenn gleich mehrere Bits in einem Statement zu manipulieren sind. Für einzelne Bits ist die obige Schreibweise "`PCMSK1.PCINT11 = 1`" bzw. "`PCMSK1.3 = 1`" wohl leserlicher. Für

```
MCUCR = MCUCR AND &B1101_1111         'Reset Bit5 in MCUCR
```

könnte dann auch geschrieben werden

`MCUCR.5 = 0` bzw. `MCUCR.BODSE = 0`, BODSE = Name des Bit #5 in MCUCR.

Das geht übrigens auch mit normalen Variablen, etwa "`bytTmp0.3 = 1`"
oder "`bytTmp0.3 = 0`": Setze Bit #3 von Byte bytTmp0 auf 1 oder 0.

Oder auch eine Abfrage ("Status" kann eine Bit- oder Byte-Variable sein, Bit reicht aber aus):

```
Status = bytTmp0.3
```

Status = 0, wenn bytTmp0 = &Bxxxx_0xxx, "x" = 0 oder 1
Status = 1, wenn bytTmp0 = &Bxxxx_1xxx

Register-Bits lassen sich entsprechend mit der Bit-Position 0 …7 oder dem Bit-Namen im Register abfragen.

Schlussfolgerungen:

- Vom Schreibaufwand halten sich beide Ansätze – BASCOM und direkter Registerzugriff - die Waage.
- Einzelne Registeroperationen machen den Code dann doch etwas transparenter. Unter Umständen können damit Funktionen realisiert werden, die mit BASCOM-Hochsprachen-befehlen so nicht direkt erreichbar sind.
- Allerdings wird dabei eine Auseinandersetzung mit den Registerbeschreibungen im jeweiligen Datenblatt abgefordert. C- und Arduino-Programmierer kommen daran ohnehin nicht vorbei, es sein denn, sie bedienen sich ungezählter Lib's.
- Register und deren Bits können sich von AVR-Familie zu AVR-Familie unterscheiden. Register mit ähnlichen Funktionen haben ggf. andere Namen, sind ggf. aufgeteilt in mehrere Unterregister, z.B. mit Index "A" oder "B", existieren hier, aber nicht dort. Entsprechend unterscheiden sich einzelne Register-Bits in Name und Funktion. Orientierungspunkt ist in jedem Falle das Datenblatt.
  Der Code wird m.E. transparenter, aber weniger portabel. Der BASCOM-Compiler wird bei direkter Registerbearbeitung möglicherweise einiges zu meckern haben, wenn er Register oder Namen der Registerbits in der Controller.def nicht findet, etwa bei einer Portierung von ATmega8 nach ATmega88.