Untersuchungen zum Zeitbedarf bei PortD.x

Häufig möchte man einzelne Bits eines Ports ansprechen, sei es lesend oder auch schreibend. Mit BASCOM geht das beim Port D ganz einfach so:

Ausgang, schreibend: `PortD.x = 1` ‘ oder auch `0`

Eingang, lesend: `Wert = PinD.x`

Dabei kann `x` für eine Konstante, aber auch für eine Variable stehen.

Die farbliche Darstellung von `PortD` und `PinD` im BASCOM-Editor suggeriert, dass es sich hier einfach um eine Register-Zuweisung handelt. Dies trifft allerdings nur zu, wenn es sich bei `x` um eine direkte **Konstante** `k` handelt, und auch da nur bei einem Schreibzugriff. BASCOM übersetzt in diesem Fall beim Kompilieren diese Zuweisung so:

`sbi portd, k` (set bit k in I/O-Register; k: Konstante zwischen 0 und 7)

bzw.

`cbi portd, k` (clear bit k in I/O-Register; k: Konstante zwischen 0 und 7)

Beim Lesen wird es schon ein wenig aufwändiger:

```
wert = 1
sbis PinD, k    (skip if bit k is set in I/O-Register; k: Konstante zwischen 0 und 7)
wert = 0
```

Wesentlich komplizierter wird es, wenn es sich bei `x` und eine Variable handelt. Hierfür gibt es bei den AVR-Mikrocontrollern nämlich keine passenden Maschinenbefehle. Dass hier wesentliche Unterschiede bestehen gegenüber dem Fall, dass es sich bei `x` um eine Konstante handelt, ist mir zum ersten Mal aufgefallen, als ich das Zeitverhalten verglichen hatte: Während die Ausführung von

`Portd.3 = 1`

nur 2 Taktzyklen beansprucht,

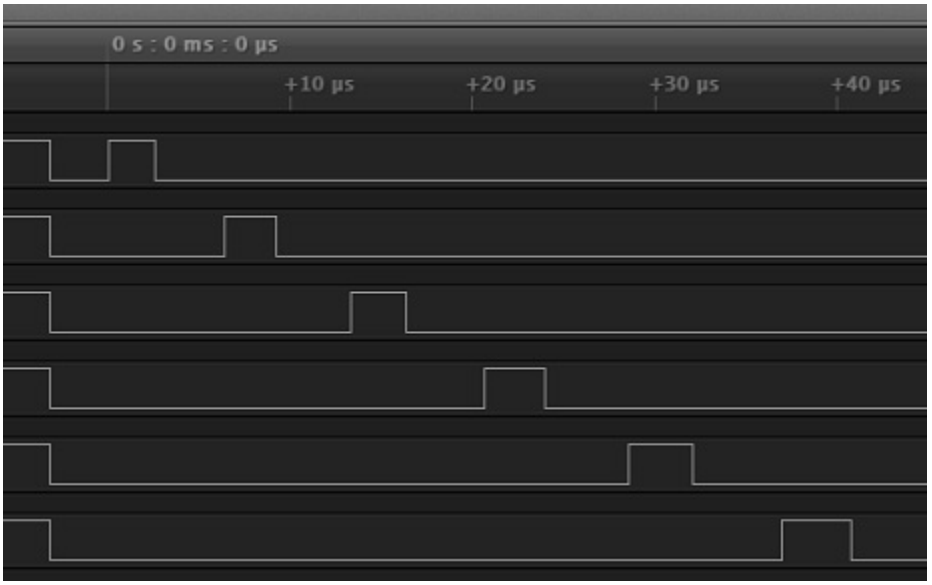
benötigt

`PortD.i = 1`

deutlich mehr Zeit, wenn es sich bei `i` um eine Variable handelt. Und mehr noch: Der Zeitbedarf steigt mit wachsendem Wert von `i`. Dies habe ich mit dem folgenden Programm getestet:

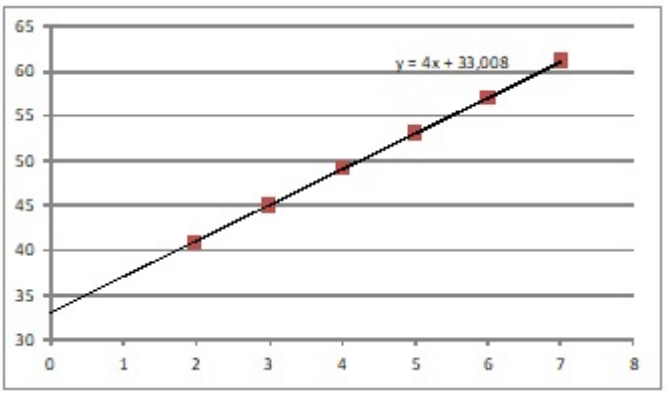
```
Dim I as byte

For I = 2 To 7
  Portd.i = 1
  Portd.i = 0
Next I
```

Die Signale an den Ports D.2 bis D.7 wurden mit einem Logik-Analysator gemessen; die negative Flanke vor der 0-Marke wurde durch `PortD = 0` vor der Schleife erzeugt.



Man sieht sofort: Die High-Phasen T_H der einzelnen Ausgänge (Diese entsprechen jeweils der Zeit für die Ausführung des Ausschaltvorgangs!) sind unterschiedlich lang und werden mit zunehmenden Wert von I größer. Ausmessen von T_H ergab:

Bit	**T_H in µs**	**Taktzyklen *Z***
2	2,563	41
3	2,813	45
4	3,063	49
5	3,313	53
6	3,563	57
7	3,813	61

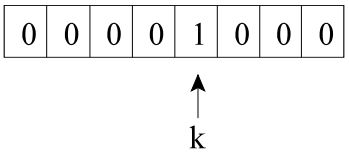


Hier ist *Z* gegen die Bit-Nummer abgetragen.

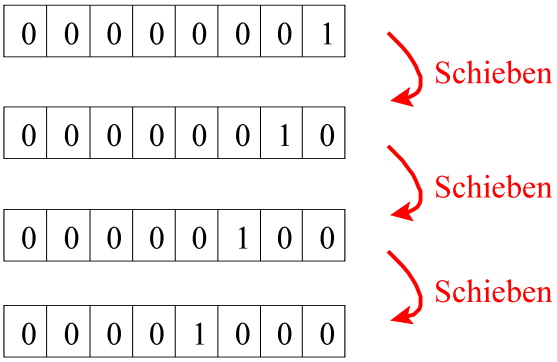
Gehen wir von einer Bit-Nummer zur nächsten über, so steigt der Zeitbedarf um einen konstanten Wert, in diesem Fall um 4 Taktzyklen.

Wie kann man dies erklären? Nun, vor einigen Jahren hatte ich einen kleinen Forth-Compiler für den Attiny2313 programmiert; dazu war es erforderlich gewesen, auch ein Assemblercode-Schnipsel für das Setzen eines variablen Bits k bei PortD zu schreiben.

Meine Idee war dabei folgende: Zunächst müssen wir ein Byte von der Art



erzeugen; dabei soll die 1 an der Position k stehen, die in einem Register (Wir nennen es k.) gespeichert ist. Dazu lädt man zunächst den Wert 1 in ein weiteres Register (Wir nennen es maske.) und schieben das Bitmuster von maske anschließend k mal (in unserem Fall also 3 mal) nach links.



Nun laden wir den Wert von PortD in ein weiteres Registers (Wir nennen es pd.). Die Werte von maske und pd verknüpfen wir durch ein logisches ODER und speichern den Wert wieder in pd ab; diesen Vorgang bezeichnet man auch als Maskieren. Das Bitmuster von pd hat dadurch den Wert 1 beim Bit k erhalten; die restlichen Bits sind unverändert geblieben. Nun wird der Wert von pd an PortD ausgegeben. Unser Ziel ist erreicht.

In der folgendenTabelle ist der Assemblercode für ein entsprechendes Programm angegeben und ausführlich erläutert. Insbesondere ist dort auch erklärt, warum das I/O-Register PortD nicht direkt mit dem maske-Register maskiert werden kann.

Mit T bezeichnen wir in der Tabelle die Anzahl der zur Ausführung eines Befehls benötigten Takte (1/2 bedeutet: 1 Takt, wenn Bedingung falsch ist, sonst 2 Takte). Aus den angegebenen Werten ergibt sich, dass bei diesem Programm ein Schleifendurchlauf 6 Takte benötigt. Das bedeutet: Der Zeitbedarf steigt bei diesem Programm um 6 Taktzyklen, wenn wir von einer Bit-Nummer zur nächst größeren übergehen.

Assembler-Code	**T**	**Kommentar**

`.def maske = r16`
`.def k = r17`
`.def pd = r18`
		In den Registern `r16`, `r17` und `r18` werden Hilfswerte sowie die Nummer `k` des Bits von PortD, das auf 1 gesetzt werden soll, gespeichert. Zur besseren Übersicht geben wir hier diesen Registern entsprechende Namen.

`in pd, portd`		Der Inhalt von `portd` wird `in` das Register `pd` kopiert. Die weiter unten benutzte Operation `or` kann nicht auf das I/O-Register `portd` angewandt werden.
`ldi maske, 1`		`ldi` *load immediate* Lade die Konstante 1 = &B00000001 in das Register `maske`
`bildemaske:`		Marke Die folgenden Befehle bis zum Befehl `rjmp bildemaske` bilden eine Schleife.
`cpi k, 0`	1	`cpi` *compare immediate* Vergleiche den Inhalt des Registers `k` mit der Konstanten 0. Sind die beiden Werte gleich, dann wird eine entsprechendes Bit im Register `SREG` gesetzt.
`breq setzen`	1/2	`breq` *branch if equal* Wenn in `SREG` festgehalten ist, dass `k` den Wert 0 hat, dann verzweigt das Programm zur Marke `setzen`, ansonsten wird der nächste Befehl ausgeführt.
`lsl maske`	1	`lsl` *logical shift left* Das Bitmuster von `maske` wird nach links geschoben. Nach dem Ausführen `lsl`-Befehls hat `maske` beim ersten Schleifendurchlauf den Wert &B00000010, beim zweiten Schleifendurchlauf den Wert &B00000100, ...
`dec k`	1	`dec` *decrement* Der Wert des Registers `k` wird um 1 erniedrigt.
`rjmp bildemaske`	2	`rjmp` *relative jump* Sprung zur Marke `bildemaske`
`setzen:`		Marke
`or pd, maske`		Die Werte von `pd` und `maske` werden (bitweise) durch ODER verknüpft und das Ergebnis &Bxxx1xxxx in pd gespeichert. Dabei steht 1 an der Position `k`.
`out portd, pd`		Der Wert von `pd` wird an PortD `ausgegeben`.