Bemerkungen zur Reihenfolge bei der Interruptbehandlung

Was geschieht, wenn mehrere Interrupt-Ereignisse gleichzeitig auftreten? Wird nur eine Interrupttroutine aufgerufen. Oder werden vielleicht alle ausgelöst, wobei man sich dann noch fragen könnte: in welcher Reihenfolge?

Mancher wird vielleicht sagen: Der Fall, dass zwei oder mehrere Interrupt-Ereignisse gleichzeitig auftreten, ist so unwahrscheinlich, dass er praktisch nicht vorkommt, und deswegen nur von theoretischem Interesse ist. Aber das stimmt nicht. Der Grund dafür wird klar, wenn man sich verdeutlicht, was für den Attiny "gleichzeitig" bedeutet.

Betrachten wir die folgende Situation: Beim Mikrocontroller tritt ein Interrupt – sagen wir vom Typ I3 – auf. Der Mikrocontroller notiert dies durch Setzen eines entsprechenden Bits, wir bezeichnen es hier als I3-Flag. Das zugehörige I3-Enable-Bit möge gesetzt sein. (Das gelte auch für die weiteren Interrupttypen, welche wir gleich noch betrachten werden, so dass wir uns um dieses Enable-Bit nicht mehr kümmern müssen.) Wenn jetzt noch das (globale) Interrupt-Bit I im Register SREG gesetzt ist, dann wird die zugehörige Interruptroutine aufgerufen. Genauer betrachtet wird der aktuelle Maschinenbefehl noch ausgeführt, dann wird die Adresse des darauf folgenden Befehls als Rücksprungadresse auf den so genannten Return-Stapel gelegt. Anschließend wird die Einsprungadresse des I3-Interrupts in den Programmzähler geladen; dort steht ein Sprungbefehl, welcher den Mikrocontroller zur Interruptroutine führt (Abb. 1). Dabei werden das I3-Flag und das I-Bit im SREG automatisch zurückgesetzt. Letzteres hat zur Folge, dass nun keine Interruptroutinen mehr aufgerufen werden können; kurz gesagt: der Mikrocontroller ist jetzt für weitere Interrupts gesperrt. Diese Sperrung wird am Ende der Interruptroutine aufgehoben. Dieses Ende wird durch einen reti-Befehl markiert. Durch diesen Befehl wird nämlich das I-Bit wieder gesetzt; gleichzeitig wird auch die Rücksprungadresse in den Programmzähler geladen, so dass der Mikrocontroller dort wieder seine Arbeit fortsetzt, wo sie ursprünglich unterbrochen worden ist.

Vektornummer	Programmadresse	Inhalt
1	$0000	springe zu <Startadresse des Hautprogramms>
2	$0001	springe zu <Startadresse der Interruptroutine I1>
3	$0002	springe zu <Startadresse der Interruptroutine I2>
4	$0003	springe zu <Startadresse der Interruptroutine I3>
...	...	...

Abb. 1: Die Interruptvektor-Tabelle

Bei Bedarf kann die Interrupt-Sperrung aufgehoben werden; dazu muss der Programmierer nur das I-Bit innerhalb der Interruptroutine wieder setzen. Nun kann die Interruptroutine ihrerseits unterbrochen worden. Weil diese Vorgehensweise zu Problemen führen kann (Insbesondere kann der Returnstapel ggf. rasch überlaufen!), wollen wir diesen Fall hier nicht weiter studieren.

Viel interessanter ist die Frage: Was geschieht mit denjenigen Interrupt-Ereignissen, welche während einer solchen Interrupt-Sperrung auftreten? Stellen wir uns vor, dass zwei weitere Interrupt-Ereignisse während der Interruptroutine auftreten; zuerst möge eine Interrupt-Ereignis vom Typ I4 auftreten und danach noch eines vom Typ I2. Wegen der Sperre werden die zugehörigen Interruptroutinen natürlich nicht direkt ausgeführt; aber die Interrupt-Ereignisse sind nicht vergessen. Durch die beiden Ereignisse sind nämlich das I2-Flag und das I4-Flag gesetzt worden. Und wenn am Ende einer Interruptroutine die Sperre durch das Setzen des I-Bits wieder aufgehoben wird, dann werden diese Flags vom Mikrocontroller abgearbeitet. Dazu laufen die gleichen Interrupt-Prozesse ab, wie sie oben beim I3-Interrupt erläutert wurden.

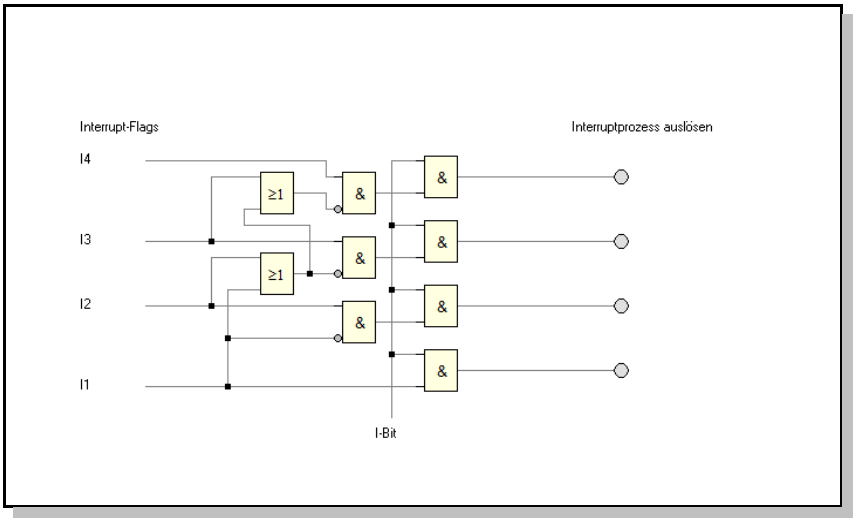

Abb. 2

Hardwaremäßig kann man sich die Situation wie in Abb. 2 vorstellen: Die Eingänge links stehen für die Flags I1 bis I4, welche durch Interrupt-Ereignisse gesetzt werden. Die UND-Schaltungen im rechten Teil der Schaltung stellen Tor-Schaltungen dar; diese lassen die Signale I1 bis I4 nur dann hindurch, wenn das I-Bit gesetzt ist. Die Ausgänge bestimmen dann, ob ein Interrupt-Prozess eingeleitet wird oder nicht.

Unklar ist in unserem Beispiel lediglich, welcher der beiden Interrupts, I2 oder I4, zuerst abgearbeitet wird. Den beiden Flags I2 und I4 ist nicht anzusehen, welches von beiden zuerst gesetzt worden ist. Ob die Flags in der einen oder anderen Reihenfolge gesetzt worden sind, ist egal: Der Mikrocontroller hat keinerlei Information darüber; die Flags könnten genauso gut auch gleichzeitig gesetzt worden sein.

In welcher Reihenfolge nun die beiden Interrupts I2 und I4 bearbeitet werden, das wird nun durch den linken Teil unserer Schaltung bestimmt: Sie legt die Priorität der einzelnen Interrupts fest. Diese Reihenfolge stimmt beim Attiny übrigens mit der Reihenfolge der Einsprungvektoren der Interrupts (Abb. 1) überein. In diesem Fall wird zuerst die I2-Routine und anschließend die I4-Routine ausgeführt, obschon die entsprechenden Interrupt-Ereignisse in umgekehrter Reihenfolge stattgefunden hatten! Dies ergibt sich unmittelbar aus der folgenden Bildserie:

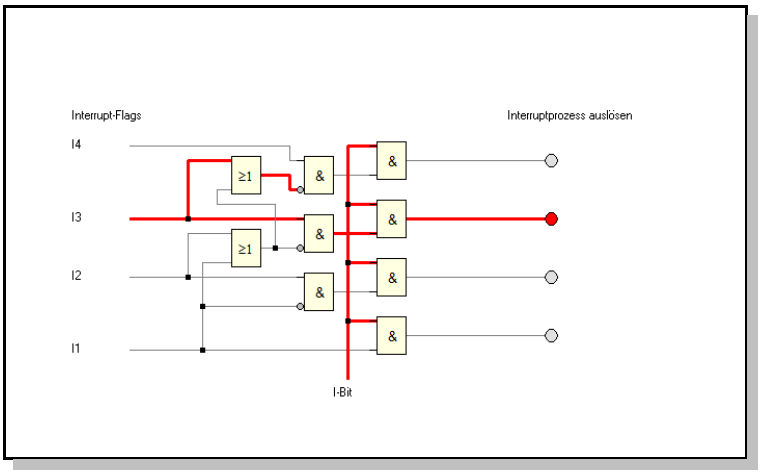

Abb. 3: Interruptprozess I3 wird ausgelöst.

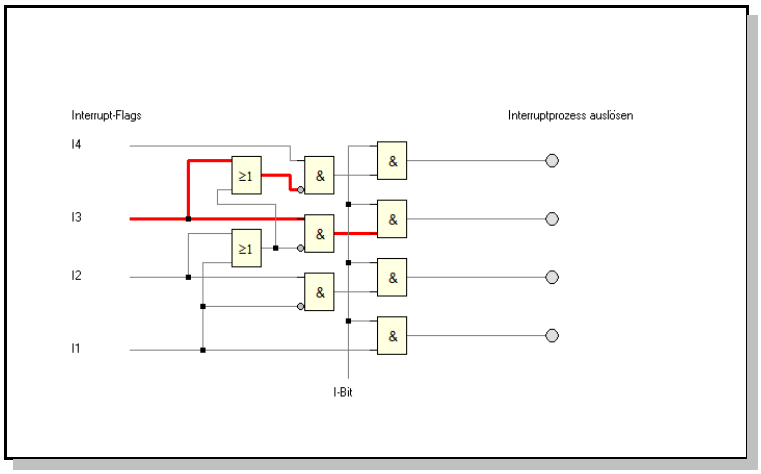

Abb. 4: Zu Beginn des I3-Prozesses wird das I-Bit zurückgesetzt...

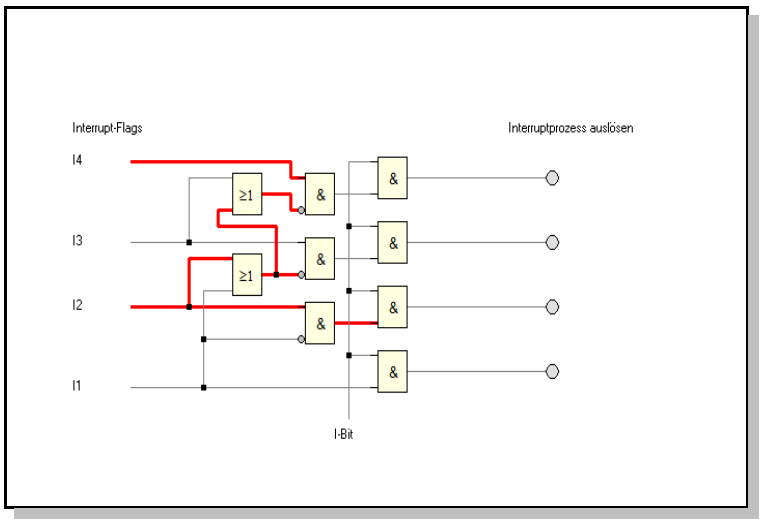

Abb. 5: ...und das I3-Flag wird zurückgesetzt. Während der I3-Interruptroutine werden die Flags I4 und I2 gesetzt.

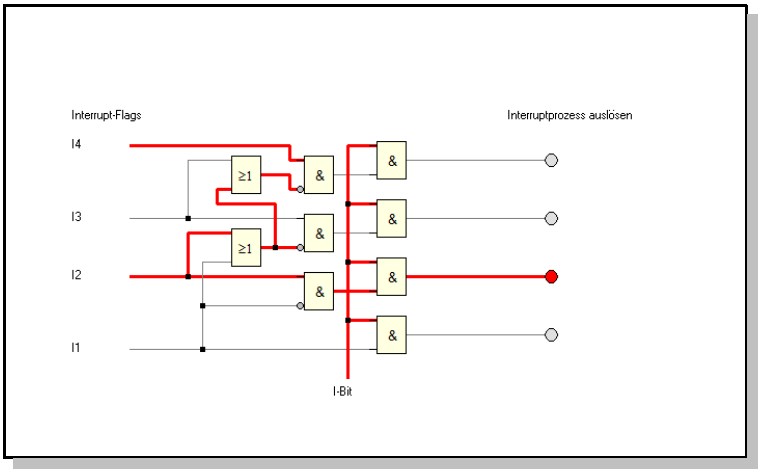

Abb. 6: Am Ende der I3-Interruptroutine wird das I-Bit gesetzt. Nun wird die I2-Interruptroutine ausgelöst.

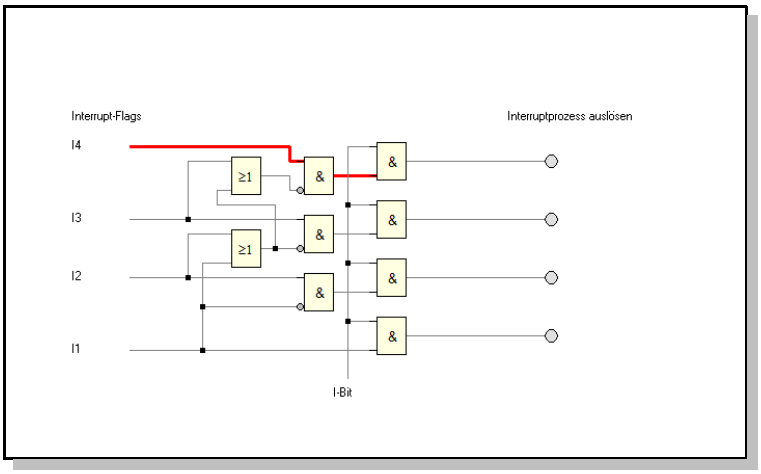

Abb. 7: Zu Beginn der I2-Routine werden das I-Bit und das I2-Flag zurückgesetzt.

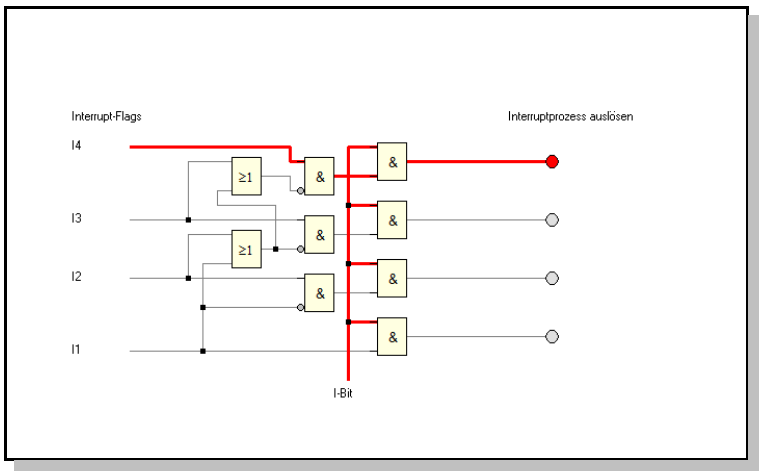

Abb. 8: Nach der I2-Routine wird die I4-Routine ausgelöst.

Fassen wir zusammen: Der Attiny kann sich in gewissem Umfang Interrupt-Ereignisse merken. Diese werde dann in der Reihenfolge der Interruptvektortabelle abgearbeitet. Treten während einer Interrupt-Sperrung allerdings mehrere Interrupt-Ereignisse des gleichen Typs auf, wird die zugehörige Routine nur ein einziges Mal ausgeführt.