

2.1 Atmega-Peripherie/Interrupts

2.1.1 Situation

Während LED2 fortlaufend blinkt, soll LED2 jederzeit sofort durch Tastendruck von T1 eingeschaltet werden können.

Dazu muss man im Programm regelmäßig nachsehen, ob die Taste gedrückt wird. Danach kann das Programm weiterlaufen. Man nennt diese Vorgehensweise *Polling*.

Leider ist das Polling sehr zeit- und programmieraufwändig. Darüberhinaus kann es passieren, dass man den richtigen Zeitpunkt der Abfrage verpasst, so dass eine Eingabe verlorenggeht. Fragt man erfahrene Entwickler wegen dieses Problems, erhält man in der Regel den Rat, *Interrupts* zu verwenden.

2.1.2 Was sind Interrupts?

Interrupts sind asynchrone Unterbrechungen des Programmablaufs eines Controllers (oder sonstigen Rechners). Sie können von außen erfolgen (externe Interrupts), durch Signale von internen Peripheriebausteinen oder durch Software verursacht werden (sogenannte Software-Interrupts). Sie werden wie Unterprogramme abgearbeitet; da sie jederzeit kommen können, erfordert ihre Abarbeitung einen etwas größeren Aufwand als Unterprogramme. Tabelle 1 zeigt, welche Interrupt-Quellen es gibt.

Nr.	Name	Auslöser
1	RESET	Reset-Eingang, Einschalten
2	INT0	Externer Interrupt 0
3	INT1	Externer Interrupt 1
4	TIMER2_COMP	Timer 2
5	TIMER2_OVF	Timer 2
6	TIMER1_CAPT	Timer 1
7	TIMER1_COMPA	Timer 1
8	TIMER1_COMPB	Timer 1
9	TIMER1_OVF	Timer 1
10	TIMER0_OVF	Timer 0
11	SPI_STC	SPI-Schnittstelle
12	USART_RX	RS232-Schnittstelle
13	USART_UDRE	RS232-Schnittstelle
14	USART_TX	RS232-Schnittstelle
15	ADC	A/D-Umsetzer
16	EE_RDY	EEPROM
17	ANA_COMP	Komparator
18	TWI	I ² C-Bus
19	SPM_RDY	Flash-EPROM (geschrieben)

Tabelle 1: Interrupt-Quellen beim ATmega8

2.1.3 Interrupt-Logik: Register und Befehle

Der Pseudo-Befehl `sei` erlaubt Interrupts, indem er das I-Bit im Statusregister SREG setzt. Im Gegensatz dazu verbietet der Pseudo-Befehl `cli` alle Interrupts, indem er das Bit I im SREG löscht. Einzige Ausnahme ist der RESET-Interrupt, der nicht verhindert (maskiert) werden kann (man nennt ihn einen *non-maskable Interrupt*).

Außerdem gibt es für jeden Interrupt ein eigenes Bit (*Maskenbit*), das die gezielte Freigabe dieses Interrupts bewirkt. Dazu gibt es spezielle Interrupt-Kontroll-Register. Ohne diese Freigabe kann der entsprechende Interrupt nicht auftreten.

Für das Auftreten des Interrupts braucht man also drei Bedingungen:

- a) Die spezielle Interruptquelle muss aktiv sein
- b) Das I-Bit muss gesetzt sein
- c) Das Maskenbit für diesen Interrupt muss gesetzt sein

2.1.4 Externe Interrupts INT0 und INT1

Das GICR-Register (*general interrupt control register*, bei manchen Controllern heißt es GIMSK) enthält die Maskenbits für die beiden externen Interrupts INT0 und INT1 (siehe Abbildung 2). INT0=1 erlaubt das Auftreten von INT0-Interrupts, INT0=0 verbietet es.

Bit	7	6	5	4	3	2	1	0
Name	INT1	INT0	-	-	-	-	IVSEL	IVCE
R/W	R/W	R/W	R	R	R	R	R/W	R/W
Default	0	0	-	-	-	-	0	0

Tabelle 2: GICR-Register

Zur weiteren Kontrolle der beiden externen Interrupts gibt es noch das MCUCR-Register (*micro controller unit control register*, Abbildung 3). ISC00 und ISC01 (Bit 0 und 1) gehören zu INT0,

Bit	7	6	5	4	3	2	1	0
Name	-	-	SE	SM	ISC11	ISC10	ISC01	ISC00
R/W	R	R	R/W	R	R	R	R/W	R/W
Default	0	0	0	0	0	0	0	0

Tabelle 3: MCUCR-Register

ISC10 und ISC11 (Bit 2 und 3) gehören zu INT1.

Die Kombinationen von Bit 0 und Bit 1 geben an, welche Flanken oder Zustände INT0 auslösen (Tabelle 4). Das Gleiche gilt für Bit 2 und Bit 3 bezüglich INT1.

ISCx1	ISCx0	Bedeutung
0	0	Interrupt bei Low-Pegel
0	1	Interrupt bei Flanken
1	0	Interrupt bei fallender Flanke
1	1	Interrupt bei steigender Flanke

Tabelle 4:

2.1.5 Interrupt-Vektor

Für jeden Interrupt, der aktiviert wurde, muss im Programm eine Adresse angegeben sein, an die im Fall des Auftretens gesprungen werden soll. Dazu dient der sogenannte *Interrupt-Vektor*. Es handelt sich in der Regel (so auch bei PC-Prozessoren) um eine Tabelle mit *Sprungadressen*. Beim ATmega ist das anders: Hier handelt es sich um eine Tabelle mit *Sprungbefehlen* zu diesen Adressen. Die Tabelle befindet sich am Beginn des Befehlsspeichers, dort, wo wir bisher die ersten Befehle unserer Programme hatten. An Adresse 0x000 liegt jetzt der Befehl für den RESET-Interrupt, der direkt nach dem Einschalten ausgeführt wird:

```

1  .include "/usr/share/avra/m8def.inc"
2  interruptvektor:
3      rjmp main      ;1      RESET      Adresse 0x000
4      rjmp intext0   ;2      INT0       Adresse 0x001

```

5	reti	;3	INT1	Adresse 0x002
6	reti	;4	TIMER2_COMP	Adresse 0x003
7	reti	;5	TIMER2_OVF	Adresse 0x004
8	reti	;6	TIMER1_CAPT	Adresse 0x005
9	reti	;7	TIMER1_COMPA	Adresse 0x006
10	reti	;8	TIMER1_COMPB	Adresse 0x007
11	reti	;9	TIMER1_OVF	Adresse 0x008
12	reti	;10	TIMER0_OVF	Adresse 0x009
13	reti	;11	SPI_STC	Adresse 0x00A
14	reti	;12	USART_RX	Adresse 0x00B
15	reti	;13	USART_UDRE	Adresse 0x00C
16	reti	;14	USART_TX	Adresse 0x00D
17	reti	;15	ADC	Adresse 0x00E
18	reti	;16	EE_RDY	Adresse 0x00F
19	reti	;17	ANA_COMP	Adresse 0x010
20	reti	;18	TWI	Adresse 0x011
21	reti	;19	SPM_RDY	Adresse 0x012
22	main:			
23	ldi r16, 0xff ;			Adresse 0x013
24	...			
25	intext0:			
26	...			

Eigentlich müssen hier nicht alle 19 Adressen stehen, sondern nur die, die auch freigegeben wurden. Der Programmstart wird durch die Marke `main:` in jedem Fall gefunden.

2.1.6 Interrupt-Handler

Wenn der externe Interrupt `INT0` auftritt, springt der Controller an die Adresse unter dem Label `intext0` (ein beliebig gewählter Name). An dieser Stelle sollte das gewünschte Unterprogramm beginnen. Man nennt es aber nicht Unterprogramm, sondern *Interrupt-Handler* oder auch *Interrupt-Service-Routine*.

1	intext0:	
2	push r16	; r16 retten
3	push r17	; r17 retten
4	in r16, SREG	; SREG nach r16
5	push r16	; und retten
6	...	; beliebige Befehle
7	...	;
8	pop r16	; vom Stack von r16
9	out SREG, r16	; und nach SREG zurueck
10	pop r17	; r17 zurueck
11	pop r16	; r16 zurueck
12	reti	; Ruecksprung

Es können darin kurze Befehlsfolgen eingebaut werden. Am Schluss des Interrupt-Handlers steht auf jeden Fall der Befehl `reti`.

Im Unterschied zu Unterprogrammen ist es bei Interrupt-Handlern zwingend notwendig, das Statusregister `SREG` zusammen mit den Registern auf dem Stack abzulegen und zum Schluss wiederzuholen. Man weiß nämlich bei Interrupt-Handlern nie, wann sie aufgerufen werden; und wenn sie gerade zwischen `dec r16` und `brne` im Hauptprogramm auftreten und dann auch noch die Status-Bits verändern, geht das Hauptprogramm in die Irre.

2.1.7 Stack und I-Flag

Damit irgendein Interrupt funktionieren kann, muss zuerst der Stackpointer initialisiert sein. Die einzige Ausnahme ist der RESET-Interrupt beim Einschalten, bei dem der Stack nicht benutzt wird.

Beim Aufruf des Interrupt-Handlers wird nämlich die Adresse des aktuell folgenden Befehls auf dem Stack abgelegt. Außerdem wird dort das I-Flag gelöscht.

Beim `reti`-Befehl wird (wie bei `ret`) der neue Wert des Programmzähler wieder vom Stack geholt und das I-Flag wird eingeschaltet, damit neue Interrupts bearbeitet werden können.

2.1.8 Beispielprogramm

```

1  .include "/usr/share/avra/m32def.inc"
2  jmp meinstart ;1: RESET Adr.0
3  jmp meinint0  ;2: INT0  Adr.2
4  ;x
5  ;#####
6  meinstart:
7      ldi r16, low(RAMEND) ; Stackpointer
8      out SPL, r16
9      ldi r16, high(RAMEND)
10     out SPH, r16
11
12     ldi r16, 0xff
13     out DDRC, r16 ; PORTC: LEDs
14     out PORTC, r16 ; alle LEDs aus
15     sbi PORTD, PD2 ; PORTD: Eingang ist Bit 2 (INT0), Pull-Up-Wid.
16 ;x
17     ldi r16, 0b01000000 ; Bit INT0 setzen, sonst nichts
18     out GICR, r16
19 ;x
20     ldi r16, 0b00000010 ; Bit ISC01=1, ISC00=0
21     out MCUCR, r16
22 ;x
23     sei ; Interrupts erlauben
24 schleife:
25     rjmp schleife ; tu nix
26 ;#####
27 meinint0: ; ISR fuer INT0 (falls Taster an PD2 gedrueckt wird):
28     cbi PORTC, PC0 ; LED0 an
29     reti
30 ;x

```

2.1.9 Verschiedenes

- Zusätzlich zum Maskenbit hat jede Interruptquelle auch ein Anzeigebit; bei INT0 und INT1 dient dazu das GIFR-Register.
- Das Anzeigebit wird beim Eintreten des Interrupt-Ereignisses gesetzt und bereits beim Sprung in den Interrupt-Handler gelöscht. Es dient dazu, anzuzeigen, ob ein noch un bearbeitetes Interrupt-Ereignis zu dieser Interruptquelle vorliegt. Sobald es null ist, kann ein neues Ereignis von derselben Quelle *bemerkt* werden. *Bearbeitet* werden kann es aber erst dann, wenn das I-Bit (wieder) gesetzt worden ist.

- Je kleiner die Nummer des Interrupts, desto höher seine Priorität. Treten zwei Interrupts gleichzeitig auf, wird der mit der höheren Priorität zuerst bearbeitet.
- Der Interrupt-Vektor hat bei den Controllern mit mehr als 8 KiB (z. B. ATmega16, ATmega32, ATmega1284) zwei statt eines 16-Bit-Wortes Platz für jeden Interrupt. Damit kann beim ATmega16 und ATmega32 der Befehl `jmp` anstatt `rjmp` benutzt werden. Das ist auch nötig, weil man dort mehr als 8 KiB Flash-EPROM (4096 Befehlsworte) besitzt. Mit dem `rjmp`-Befehl kann man aber maximal 4096 Adressen weit springen, so dass er dort nicht ausreicht. Bei diesen Controllern ersetzt man also in der Tabelle jeden `rjmp`-Befehl durch einen `jmp`-Befehl.

Jeder fehlende Sprungbefehl muss dann durch zwei Befehlsworte ersetzt werden. Nach jedem `reti`-Befehl muss also ein `nop`-Befehl folgen (siehe Beispiel unten).

2.1.10 PCINT-Interrupts beim ATmega1284p

Bei den neueren ATmega-Controllern (z. B. 324, 644 und 1284p) gibt es neben den genannten drei Interrupt-Quellen noch vier weitere. Sie heißen *Pin Change Interrupts* (PCI) und zeigen eine Taktflanke an einem Eingang an. Ein Nachteil dabei ist, dass man nicht einschränken kann, welche Taktflanke man meint. Ein Vorteil liegt darin, dass man jeden Ein-/Ausgabe-Anschluss für einen Interrupt verwenden kann.

Das Ganze funktioniert so: Je acht Anschlüsse sind einem Interrupt zugeordnet (siehe Tabelle 5). Die Namen PCINT0 bis PCINT31 suggerieren, dass es sich um 32 verschiedene Interrupts

Anschlüsse	Einzel-Interrupts	Interrupt-Quelle
PORTA 7 bis PORTA 0	PCINT7 bis PCINT0	PCI0
PORTB 7 bis PORTB 0	PCINT15 bis PCINT8	PCI1
PORTC 7 bis PORTC 0	PCINT23 bis PCINT16	PCI2
PORTD 7 bis PORTD 0	PCINT31 bis PCINT24	PCI3

Tabelle 5: Zuordnung Anschluss–Interrupt–Quelle

handelt. Man hat aber nur vier Quellen, die man voneinander unterscheiden kann. Fragt man mehrere Anschlüsse von PORTA gleichzeitig ab, muss man im Interrupt-Handler selbst unterscheiden, welcher Anschluss den Interrupt ausgelöst hat.

Die Konfiguration funktioniert wie folgt:

- Man gibt im PCICR-Register an, welchen (oder welche) der Quellen man benutzen möchte. Jedes der Bits PCIE0 bis PCIE3 dient als Freigabe (*enable*) für die zugehörige Quelle.
- Nun muss man noch in den Registern PCMSK0 bis PCMSK3 angeben, welcher Anschluss (oder welche Anschlüsse) zur Interrupt-Quelle durchgeschaltet wird. Für jeden Anschluss ist dabei ein eigenes Bit vorhanden.
- Wie immer muss zum Schluss das I-Flag gesetzt werden.

Tabelle 6 zeigt die entsprechenden Register. Hier ist ein Beispiel, bei dem nach Eintreffen eines PCI2-Interrupts am Anschluss PC7 (=PCINT23) die LED an PB0 eingeschaltet wird:

```

1 .include "/usr/share/avra/m1284pdef.inc"
2 ; Pegel–Änderung an PC7 soll per Interrupt LED0 einschalten
3 ; uC: ATmega1284p ; LEDs: PORTB; Schalter: PORTC
4 tabelle:
5     jmp main          ;1 RESET
6     reti              ;2 INT0 an PD2
7     nop
8     reti              ;3 INT1 an PD3

```

```

9      nop
10     reti          ;4 INT2 an PB2
11     nop
12     reti          ;5 PCINT0 an PA0-PA7
13     nop
14     reti          ;6 PCINT1 an PB0-PB7
15     nop
16     jmp tast_uprog ;7 PCINT2 an PC0-PC7
17 main:
18     ldi r16, low(RAMEND) ; Stackpointer
19     out SPL, r16
20     ldi r16, high(RAMEND)
21     out SPH, r16
22     ldi r16, 0xff      ; LEDs als Ausgaenge
23     out DDRB, r16
24 ;
25     ; Teil(e) von PORTC als PCINT verwenden
26     ldi r16, 1<<PCIE2 ; deshalb PCIE2-Bit setzen
27     sts PCICR, r16     ; in SRAM-Reg. PCICR speichern
28     ; PORTC: Eingang ist Bit 7 (PCINT23)
29     ldi r16, 1<<PCINT23 ; Bit 7 setzen, sonst nichts
30     sts PCMSK2, r16    ; In SRAM-Register PCMSK2 speichern
31     sei               ; Interrupts erlauben
32 ;
33 schleife:
34     rjmp schleife      ; tu nix
35 tast_uprog:           ; falls PCINT23 ausloest:
36     cli
37     sbi PORTB, PB0 ; LED0 an
38     reti

```

Name	Bit	7	6	5	4	3	2	1	0	Adresse
PCICR	Bit	–	–	–	–	PCIE3	PCIE2	PCIE1	PCIE0	SRAM 0x68
	R/W	R	R	R	R/W	R/W	R/W	R/W	R/W	
	Init.	0	0	0	0	0	0	0	0	
	Bit	7	6	5	4	3	2	1	0	
PCMSK3		PCINT31	PCINT30	PCINT29	PCINT28	PCINT27	PCINT26	PCINT25	PCINT24	SRAM 0x73
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	Init.	0	0	0	0	0	0	0	0	
	Bit	7	6	5	4	3	2	1	0	
PCMSK2		PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	SRAM 0x72
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	Init.	0	0	0	0	0	0	0	0	
	Bit	7	6	5	4	3	2	1	0	
PCMSK1		PCINT15	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	SRAM 0x71
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	Init.	0	0	0	0	0	0	0	0	
	Bit	7	6	5	4	3	2	1	0	
PCMSK0		PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	SRAM 0x70
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	Init.	0	0	0	0	0	0	0	0	
	Bit	7	6	5	4	3	2	1	0	
PCIFR		–	–	–	–	PCIF3	PCIF2	PCIF1	PCIF0	SFR 0x1B
	R/W	R	R	R	R	R/W	R/W	R/W	R/W	
	Init.	0	0	0	0	0	0	0	0	

Tabelle 6: Register für PCI0 bis PCI3