Interruptsystem AVR (ATMEL328p)

Table of Contents

Interruptsystem AVR (ATMEL328p)..1
Voraussetzungen, dass ein Interrupt ausgelöst wird:..1
Steuer-Register..1
Stack..2
Das Interrupt-System...2
Interrupt Vector Table (IVT)..3
Der Request...3
Externe Interrupts INT0, INT1...4
Pin Change Interrupt..4
Software-Interrupt..4
Programmierung...5
Arduino...5
LIBC..7
Anhang..8
Interrupt Vector Table...8

Interrupt = automatische Unterbrechung eines laufenden Programmes durch ein Gerät und Fortsetzung nachdem eine Interrupt-Behandlungs-Routine (ISR) abgearbeitet wurde, ohne dass das Hauptprogramm etwas davon merkt. Man kann allerdings über globale Variablen dem Hauptprogramm Informationen schicken.

Die ISR steht auf einer fest eingebauten Adresse (= Interrupt Vector Address) am Beginn des Programmspeichers.

Voraussetzungen, dass ein Interrupt ausgelöst wird:

1. Anfrage (Interrupt Request)

2. Erlaubnis (Interrupt Enable)

3. Priorität (interrupt Prio entweder über einen Controller konfigurierbar oder fix eingebaut)

4. Interruptsystem einschalten (sei(); //set interrupt flag)

Steuer-Register

am Beispiel eines externen Interrupts der durch eine Flanke am Pin PD2 des Prozessors ausgelöst wird:

Anfrage: FLAG Register (EIFR external int flag reg)

Erlaubnis: MASK Register (EIMSK ext int mask reg)

Priorität: eingebaut, kein Steuer-Register; je kleiner die Einsprung-Adresse, desto höher die Priorität

Interrupt-System einschalten: SREG Register (status reg) enthält das I-Flag, wenn es gesetzt ist sind alle Interrupts aktiviert

Stack

Der Stack ist ein logischer Speicher der am oberen Ende des internen SRAM angelegt wird. Er wird durch den Stack-Pointer verwaltet.

Der Stack-Pointer zeigt immer an das obere Ende des Stapels wo die nächsten Daten abgelegt bzw. geholt werden können. Nach dem Schreiben auf den Stapel wird die Adresse automatisch weitergestellt, so dass durch aufeinander folgende PUSH Befehle sehr einfach und schnell Daten auf den Stack gelegt werden können. Ein POP Befehl liest die Daten dann in umgekehrter Reihenfolge wieder aus.

Beim Programmstart muss der Stack-Pointer softwaremäßig auf das Speicherende gestellt werden (macht der C-Compiler automatisch).

Das Interrupt-System

- Interrupt-Quellen: Extern, Timer, ADC, USART usw. (siehe IntVectorTable im Anhang)
- jede I-Quelle braucht ein zugehöriges Enable-Flag
- Request Flag wird durch das Gerät gesetzt; ob die Anfrage einen Interrupt auch auslöst hängt davon ab, ob dieser Interrupt enabled ist und ob die Priorität hoch genug ist
- die Rücksprungadresse muss gesichert werden; wenn das Hauptprogramm unterbrochen wird, muss die Adresse des nächsten Befehls gerettet werde; dies erfolgt, indem die Adresse auf den Stack gerettet wird
- Interrupt-Kontrollregister im IO Space
- Global Interrupt Enable Bit (I-Flag) liegt im Status Register SREG
- Die Interrupt-Service-Routine darf keine Veränderungen an den Registern hinterlassen, auch SREG wird nicht automatisch gesichert, vor dem Rücksprung muss der Status des Hauptprogramms wiederhergestellt werden (macht der C Compiler automatisch)
- Mehrere Requests gleichzeitig: beim Aussprung aus ISR wird immer min. 1 Zyklus im Hauptprogramm abgearbeitet, bevor ein weiterer I ausgeführt wird;

C-Compiler

Der C-Compiler übernimmt das Retten der Register des Hauptprogramms

- nach sei() wird noch ein Befehl abgearbeitet, bevor anstehende Interrupts verarbeitet werden, cli() wirkt schlagartig
- Reaktionszeit auf einen Interrupt: Einsprung 4 Zyklen (Speichern der Rücksprungadresse, löschen des I Flags); Rücksprung auch 4 Zyklen
- SREG:I wird automatisch resetiert, wenn in die ISR gesprungen wird; wird automatisch

wieder gesetzt, wenn aus der ISR zurückgesprungen wird. Im folgenden Code:

```
ISR(TIMER2_OVF_vect){
 cli();
}
```

… würde also das i Flag automatisch wieder gesetzt

- sei() innerhalb einer ISR → verschachtelte Interrupt-Aufrufe (jeder Interrupt darf die ISR unterbrechen); es gehen jedoch ein paar Zyklen verloren bis das sei() wirkt; braucht man es schneller, dann mit **ISR(INT0_vect,ISR_NOBLOCK)** {…}
- Interrupt request flag **löschen**: eine **1 schreiben**
  - `TIFR |= _BV(TOV0); /* wrong! */  simply use  TIFR = _BV(TOV0);`
  - warum falsch: weil es länger dauert und TIFR = 00000010 die anderen Bits ohnehin nicht verändert (sie werden ja durch eine 0 nicht gelöscht)

Interrupt Vector Table (IVT)

- Jede Interrupt-Quelle hat eine eigene Programmspeicher-Adresse (= Interrupt Vector)
- Priorität geregelt nach der Anordnung innerhalb der IVT (Reset hat höchste Prio)

Table 12-6. Reset and Interrupt Vectors in ATmega328 and ATmega328P

VectorNo.	Program Address[2]	Source	Interrupt Definition
1	0x0000[1]	RESET	External Pin, Power-on Reset, Brown-out
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	P	
7	0x000C	V	
8	0x000E	T	

```
Address  Labels  Code         Comments
0x0000           jmp  RESET     ; Reset Handler
0x0002           jmp  EXT_INT0  ; IRQ0 Handler
0x0004           jmp  EXT_INT1  ; IRQ1 Handler
0x0006           jmp  PCINT0    ; PCINT0 Handler
```

- die Interrupts können an den Anfang des BOOT-Bereichs verschoben werden (IVSEL Interrupt-Vector-Select-Flag oder BOOTRST Boot-Reset-Fuse)
  - IVSEL … bewegt die IVT in Bootbereich, Reset-Adresse bleibt 0
  - BOOTRST … gesamte IVT (inkl. Reset) in Bootbereich
- werden die Interrupts nicht benötigt, so kann der Speicherbereich der I-Vectortable für Programmcode genutzt werden

Der Request

1. **Trigger durch Request-Flag**
   - Hardware resetiert das Request Flag
   - Software kann das Request Flag auch löschen (Schreiben einer „1“)
   - Request ohne Enable bleibt bestehen, bis Enable aktiviert oder Software-Reset
   - stehen mehrere Requests an, entscheidet die eingebaute Priorität

- Einsprung I-Vektor

2. **Trigger ohne Request Flag**
   solange die Interrupt-Bedingung erfüllt ist, bleibt die Anfrage, verschwindet der Request vor einem Enable, dann wird kein Interrupt ausgelöst

   z.B: level triggered external interrupt (EICRA: ISC01/ISC00=0/0 The low level Interrupt)

Externe Interrupts INT0, INT1

- Modus: steigende Flanke, fallende Flanke, Pegel-Änderung und LOW Pegel löst aus
- EICRA external int control register A (MODUS)
- EIMSK – External Interrupt Mask Register (ENABLE)
- EIFR – External Interrupt Flag Register (REQUEST)
- LOW-Pegel Interrupt und Flankenerkennung-Interrupt arbeiten asynchron (wichtig für Aufwecken aus Sleep-Modi, weil dann evtl. kein Takt vorhanden ist)

Pin Change Interrupt

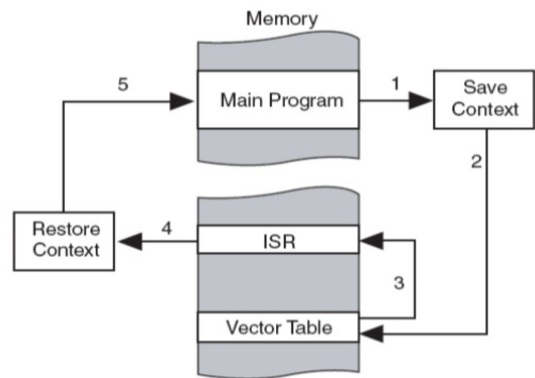
- PCINT2 … (pin change interrupt 2); wird getriggert, wenn einer der Pins PCINT 23..16 aus der Pin-Gruppe 2 toggelt
- PCINT1 … Pin-Gruppe 1; Pins PCINT 14..8
- PCINT0 … Pin-Gruppe 2; Pins PCINT 7..0
- Pin-Change Interrupts arbeiten asynchron, benötigen also keinen Takt, um aus einem Schlafmodus den uC aufzuwecken
- PCICR (interrupt control reg); erlaubt eine der Pin-Gruppen
- PCMSK2,PCMSK1,PCMSK0 (mask register) erlaubt die einzelnen Pins einer Pin-Gruppe
- PCIFR – (Flag Register) enthält für jede Pin-Gruppe das Request-Flag

Software-Interrupt

- "Interrupt" kann durch ein Programm ausgelöst werden; dies wird benutzt, um einen Unterprogrammaufruf in einem anderen Kontext (z.B. Betriebssystem, Taskwechsel) zu machen.
- ist bei AVR nicht sinnvoll; aber ein externer Interrupt bzw. ein Pin-Change Interrupt wird auch ausgelöst, wenn die Pins als Outputs geschaltet sind und z.B: eine Flanke programmiert wird

Programmierung

- Abarbeitung eines Interrupts:
  - Request
  - Einsprung in den I-Vector (Rücksprungadresse automatisch auf den Stack)
  - Start der ISR (Status retten, Register retten, abarbeiten, Register und Status zurückladen, reti)
  - Rücksprung aus der ISR (Rücksprungadresse automatisch vom Stack)



- das Hauptprogramm darf von der Unterbrechung nichts mitbekommen, daher Status retten und wiederherstellen
- in der ISR ist globales Intflag (SREG:I) automatisch gelöscht; wird nach dem Rücksprung wieder hergestellt
- Regel: ISR so schnell wie möglich verlassen
- Datenaustausch mit main() über globale Variable
- Unterbrechung verhindern: I-Flag löschen (cli)
- der C Compiler löscht beim Eintritt in die ISR das I-Flag und setzt es bei RETI wieder; das Status-Register und andere Register werden automatisch gesichert
- globale Variable, die innerhalb der ISR verarbeitet werden, sollten als **`volatile`** gekennzeichnet werden. Der Compiler erkennt nicht, dass eine Variable innerhalb einer ISR verändert wird und würde sie daher u.U. wegoptimieren.

Arduino

Eine leere Arduino Anwendung verwendet bereits das Interrupt-System!

- **I-Flag ist gesetzt**
- Timer 0 läuft + Timer overflow Interrupt enabled
- Timer 1 läuft, kein Interrupt
- Timer 2 läuft, kein Interrupt

Achtung! Innerhalb einer Int-Service-Routine funktionieren **delay**(), **micros**() und **millis**() nicht richtig, da diese Funktionen interrupt-basiert sind.

`delayMicroseconds()` benutzt keine Zähler und wird deshalb normal funktionieren.

Beispiele

Bsp1

```
#include <Arduino.h>
void loop(void );
void isr_INT0();

volatile uint8_t num_of_interrupts;

void loop(void)
{ setup();
  sei();
  while(1)
  {
    Serial.println(num_of_interrupts, DEC);
    delay(1000);
  }
}

void isr_INT0(){
  if (digitalRead(13)==HIGH)
     digitalWrite(13,LOW);
       else digitalWrite(13,HIGH);
  num_of_interrupts++;
}

void setup(){
  Serial.begin(57600);
  pinMode(13,OUTPUT);
  pinMode(2,INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(INT0),isr_INT0,FALLING); //PIN 2
}
```

"Volatile" verhindert, dass die Variable beim Compilieren wegoptimiert wird; dies kann man überprüfen, wenn man das Schlüsselwort volatile entfernt und in der Perspektive "release" compiliert. Man kriegt keine Fehlermeldung, aber die Zählfunktion funktioniert nicht.

LIBC

Es wird ein Arduino-Projekt angelegt, um die Lib HardwareSerial zur Verfügung zu haben.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <HardwareSerial.h>
#include <util/delay.h>

void setup();

uint8_t num_of_interrupts;

int main(void)
{
	setup();
	sei();
	while(1)
	{
		Serial.println(num_of_interrupts, DEC);
		//_delay_ms(1000);
	}
}

ISR(INT0_vect){
	num_of_interrupts++;
}

void setup(){
	Serial.begin(57600);
	EIMSK |= _BV(INT0);  //enable
	EICRA |= _BV(ISC01); //falling edge
}
```

ISR(BADISR_vect) … wenn keine ISR vorhanden, dann springe hierher

ISR_NAKED … die Kontrolle über die ISR in der eigenen Hand (SREG und RETI())

ISR(INT0_vect, ISR_BLOCK) // das Gleiche, wie ISR(INT0_vect) d.h. während der Ausführung ist i-Flag gelöscht und wird beim Aussprung aus der ISR wieder gesetzt.

ISR_NOBLOCK .. an den Anfang der ISR wird vom Compiler sei() gesetzt, so dass die ISR unterbrochen werden kann.

ISR_ALIAS(INT1_vect, INT0_vect); //beide tun das Gleiche

EMPTY_INTERRUPT() aus dem Schlafmodus aufwecken, aber keine Funktionalität

Anhang

Interrupt Vector Table

http://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html

Vector name	Description
ADC_vect	ADC Conversion Complete
ANALOG_COMP_0_vect	Analog Comparator 0
ANALOG_COMP_1_vect	Analog Comparator 1
ANALOG_COMP_2_vect	Analog Comparator 2
ANALOG_COMP_vect	Analog Comparator
ANA_COMP_vect	Analog Comparator
CANIT_vect	CAN Transfer Complete or Error
EEPROM_READY_vect	
EE_RDY_vect	EEPROM Ready
EE_READY_vect	EEPROM Ready
EXT_INT0_vect	External Interrupt Request 0
INT0_vect	External Interrupt 0
INT1_vect	External Interrupt Request 1
INT2_vect	External Interrupt Request 2
INT3_vect	External Interrupt Request 3
INT4_vect	External Interrupt Request 4
INT5_vect	External Interrupt Request 5
INT6_vect	External Interrupt Request 6
INT7_vect	External Interrupt Request 7
IO_PINS_vect	External Interrupt Request 0
LCD_vect	LCD Start of Frame
LOWLEVEL_IO_PINS_vect	Low-level Input on Port B
OVRIT_vect	CAN Timer Overrun
PCINT0_vect	Pin Change Interrupt Request 0
PCINT1_vect	Pin Change Interrupt Request 1
PCINT2_vect	Pin Change Interrupt Request 2
PCINT3_vect	Pin Change Interrupt Request 3
PCINT_vect	
PSC0_CAPT_vect	PSC0 Capture Event
PSC0_EC_vect	PSC0 End Cycle
PSC1_CAPT_vect	PSC1 Capture Event
PSC1_EC_vect	PSC1 End Cycle
PSC2_CAPT_vect	PSC2 Capture Event
PSC2_EC_vect	PSC2 End Cycle
SPI_STC_vect	Serial Transfer Complete

SPM_RDY_vect	Store Program Memory Ready
SPM_READY_vect	Store Program Memory Read
TIM0_COMPA_vect	Timer/Counter Compare Match A
TIM0_COMPB_vect	Timer/Counter Compare Match B
TIM0_OVF_vect	Timer/Counter0 Overflow
TIM1_CAPT_vect	Timer/Counter1 Capture Event
TIM1_COMPA_vect	Timer/Counter1 Compare Match A
TIM1_COMPB_vect	Timer/Counter1 Compare Match B
TIM1_OVF_vect	Timer/Counter1 Overflow
TIMER0_CAPT_vect	ADC Conversion Complete
TIMER0_COMPA_vect	TimerCounter0 Compare Match A
TIMER0_COMPB_vect	Timer Counter 0 Compare Match B
TIMER0_COMP_A_vect	Timer/Counter0 Compare Match A
TIMER0_COMP_vect	Timer/Counter0 Compare Match
TIMER0_OVF0_vect	Timer/Counter0 Overflow
TIMER0_OVF_vect	Timer/Counter0 Overflow
TIMER1_CAPT1_vect	Timer/Counter1 Capture Event
TIMER1_CAPT_vect	Timer/Counter Capture Event
TIMER1_CMPA_vect	Timer/Counter1 Compare Match 1A
TIMER1_CMPB_vect	Timer/Counter1 Compare Match 1B
TIMER1_COMP1_vect	Timer/Counter1 Compare Match
TIMER1_COMPA_vect	Timer/Counter1 Compare Match A
TIMER1_COMPB_vect	Timer/Counter1 Compare MatchB
TIMER1_COMPC_vect	Timer/Counter1 Compare Match C
TIMER1_COMPD_vect	Timer/Counter1 Compare Match D
TIMER1_COMP_vect	Timer/Counter1 Compare Match
TIMER1_OVF1_vect	Timer/Counter1 Overflow
TIMER1_OVF_vect	Timer/Counter1 Overflow
TIMER2_COMPA_vect	Timer/Counter2 Compare Match A
TIMER2_COMPB_vect	Timer/Counter2 Compare Match A
TIMER2_COMP_vect	Timer/Counter2 Compare Match
TIMER2_OVF_vect	Timer/Counter2 Overflow
TIMER3_CAPT_vect	Timer/Counter3 Capture Event
TIMER3_COMPA_vect	Timer/Counter3 Compare Match A
TIMER3_COMPB_vect	Timer/Counter3 Compare Match B
TIMER3_COMPC_vect	Timer/Counter3 Compare Match C
TIMER3_OVF_vect	Timer/Counter3 Overflow
TIMER4_CAPT_vect	Timer/Counter4 Capture Event
TIMER4_COMPA_vect	Timer/Counter4 Compare Match A
TIMER4_COMPB_vect	Timer/Counter4 Compare Match B
TIMER4_COMPC_vect	Timer/Counter4 Compare Match C
TIMER4_OVF_vect	Timer/Counter4 Overflow
TIMER5_CAPT_vect	Timer/Counter5 Capture Event

TIMER5_COMPA_vect	Timer/Counter5 Compare Match A
TIMER5_COMPB_vect	Timer/Counter5 Compare Match B
TIMER5_COMPC_vect	Timer/Counter5 Compare Match C
TIMER5_OVF_vect	Timer/Counter5 Overflow
TWI_vect	2-wire Serial Interface
TXDONE_vect	Transmission Done, Bit Timer Flag 2 Interrupt
TXEMPTY_vect	Transmit Buffer Empty, Bit Itmer Flag 0 Interrupt
UART0_RX_vect	UART0, Rx Complete
UART0_TX_vect	UART0, Tx Complete
UART0_UDRE_vect	UART0 Data Register Empty
UART1_RX_vect	UART1, Rx Complete
UART1_TX_vect	UART1, Tx Complete
UART1_UDRE_vect	UART1 Data Register Empty
UART_RX_vect	UART, Rx Complete
UART_TX_vect	UART, Tx Complete
UART_UDRE_vect	UART Data Register Empty
USART0_RXC_vect	USART0, Rx Complete
USART0_RX_vect	USART0, Rx Complete
USART0_TXC_vect	USART0, Tx Complete
USART0_TX_vect	USART0, Tx Complete
USART0_UDRE_vect	USART0 Data Register Empty
USART1_RXC_vect	USART1, Rx Complete
USART1_RX_vect	USART1, Rx Complete
USART1_TXC_vect	USART1, Tx Complete
USART1_TX_vect	USART1, Tx Complete
USART1_UDRE_vect	USART1, Data Register Empty
USART2_RX_vect	USART2, Rx Complete
USART2_TX_vect	USART2, Tx Complete
USART2_UDRE_vect	USART2 Data register Empty
USART3_RX_vect	USART3, Rx Complete
USART3_TX_vect	USART3, Tx Complete
USART3_UDRE_vect	USART3 Data register Empty
USART_RXC_vect	USART, Rx Complete
USART_RX_vect	USART, Rx Complete
USART_TXC_vect	USART, Tx Complete
USART_TX_vect	USART, Tx Complete
USART_UDRE_vect	USART Data Register Empty
USI_OVERFLOW_vect	USI Overflow
USI_OVF_vect	USI Overflow
USI_START_vect	USI Start Condition
USI_STRT_vect	USI Start
USI_STR_vect	USI START
WATCHDOG_vect	Watchdog Time-out

WDT_OVERFLOW_vect	Watchdog Timer Overflow
WDT_vect	Watchdog Timeout Interrupt