U3 Interruptbehandlung und Stromsparmodi

- Wiederholung: Interrupts
- Wiederholung: Stromsparmodi des AVR
- Aufgabe 4: Ampelsteuerung

U3-1 Externe Interrupts des AVR-µC

1 Flanken-/Pegel-Steuerung

- Externe Interrupts durch Pegel(änderung) an bestimmten I/O-Pins
  - ATmega32: 3 Quellen an den Pins PD2, PD3 und PB2
- Pegel- oder flanken-gesteuert
  - ➤ Abhängig von der jeweiligen Interruptquelle
- Beispiel: Externer Interrupt 2 (INT2)

ISC2	IRQ bei:
0	Fallender Flanke
1	Steigender Flanke

- Dokumentation im ATmega32-Datenblatt
  - Interruptbehandlung allgemein: S. 44-48
  - Externe Interrupts: S. 66-68

1 Flanken-/Pegel-Steuerung (2)

- Beim ATmega32 befinden sich die ISC-Bits im
  MCU Control Register (`MCUCR`) und
  MCU Control and Status Register (`MCUCSR`)
- Position der ISC-Bits in den Registern durch Makros definiert
  `ISCn0` und `ISCn1 (INT0 und INT 1) oder ISCn (INT2)`
- Beispiel: `INT2` bei ATmega32 für fallende Flanke konfigurieren

```
/* die ISCs für INT2 befinden sich im MCUCSR */
MCUCSR &= ~(1<<ISC2);     /* ISC2 löschen */
```


2 Maskieren

- Alle Interruptquellen können separat ausgeschaltet (=maskiert) werden
- Für externe Interrupts ist folgendes Register zuständig:
  - ATmega32: General Interrupt Control Register (`GICR`)
- Die Bitpositionen in diesem Register sind durch Makros `INTn` definiert
- Ein gesetztes Bit aktiviert den jeweiligen Interrupt
- Beispiel: Interrupt 2 aktivieren

```
GICR |= (1<<INT2);      /* demaskiere Interrupt 2 */
```

2 Maskieren (2)

- *Alle* Interrupts können nochmals bei der CPU direkt abgeschaltet werden
  - ➤ durch speziellen Maschinenbefehl `cli`
- Die Bibliothek avr-libc bietet hierfür Makros an:
  (`#include <avr/interrupt.h>`)
  - ◆ `sei()` - lässt Interrupts zu
  - ◆ `cli()` - blockiert alle Interrupts
- Beispiel

```
#include <avr/interrupt.h>

sei();                    /* IRQs zulassen */
```

- Innerhalb eines Interrupt-Handlers sind automatisch alle Interrupts blockiert, beim Verlassen werden sie wieder deblockiert
- Beim Start des µC sind die Interrupts bei der CPU abgeschaltet

3 Interrupt-Handler

- Installieren eines Interrupt-Handlers wird durch C-Bibliothek unterstützt
- Makro `ISR` (Interruptvektor) zur Definition einer Handler-Funktion (`#include <avr/interrupt.h>`)
- Als Parameter gibt man dem Makro den gewünschten Vektor an, z. B. `INT2_vect` für den externen Interrupt 2
  - ◆ verfügbare Vektoren: siehe avr-libc-Doku zu `avr/interrupt.h`
  - ◆ verlinkt im Doku-Bereich auf der SPiC-Webseite
- Beispiel: Handler für Interrupt 2 implementieren

```
#include <avr/interrupt.h>
static int zaehler;

ISR (INT2_vect) {
  zaehler++;
}
```


4 Implementierung von Interruptbehandlungen

- Während einer Interruptbehandlung sind andere Interrupts gesperrt
- Auftreten eines Interrupts wird durch Flag in Statusregister vermerkt
  - ◆ Dieses Flag (Bit) ist entweder 0 oder 1
  - ◆ es kann also maximal ein Interrupt zwischengespeichert werden
  - ◆ weitere Interrupts während einer Interruptsperre gehen verloren
- Gleiches gilt für mit Interruptsperren synchronisierte kritische Abschnitte
- Das Problem ist generell nicht zu verhindern
  ☞ Risikominimierung: Interruptbehandlungen sollten möglichst kurz sein
- `sei` sollte niemals in einer Interruptbehandlung ausgeführt werden
  - ◆ potentiell endlos geschachtelte Interruptbehandlung
  - ◆ Stackoverflow möglich

U3-2 Stromsparmodi von AVR-Prozessoren

- AVR-basierte Geräte oft batteriebetrieben (z.B. Sensorknoten)
- Energiesparen kann die Lebensdauer drastisch erhöhen
- AVR-Prozessoren unterstützen unterschiedliche Powersave-Modi
  - ➤ Deaktivierung funktionaler Einheiten
  - ➤ Unterschiede in der "Tiefe" des Schlafes
  - ➤ Nur aktive funktionale Einheiten können die CPU aufwecken
  - ➤ Standard-Modus: Idle
- In tieferen Sleep-Modi wird der I/O-Takt deaktiviert
  - ➤ nur asynchron arbeitende Einheiten können die CPU aufwecken
  - ➤ low-level-gesteuerte externe Interrupts werden asynchron ermittelt
  - ➤ flankengesteuerte Interrupts benötigen evtl. den Taktgeber
- Dokumentation im ATmega32-Datenblatt, S. 32-35

1 Nutzung der Sleep-Modi

- Unterstützung aus der avr-libc:
  (`#include <avr/sleep.h>`)
  - `sleep_enable()` - aktiviert den Sleep-Modus
  - `sleep_cpu()` - setzt das Gerät in den Sleep-Modus
  - `sleep_disable()` - deaktiviert den Sleep-Modus
  - `set_sleep_mode(uint8_t mode)` - stellt den zu verwendenden Modus ein
- Dokumentation von `avr/sleep.h` in avr-libc-Dokumentation
  ☞ verlinkt im Doku-Bereich auf der SPiC-Webseite
- Beispiel

```
#include <avr/sleep.h>
set_sleep_mode(SLEEP_MODE_IDLE); /* Idle-Modus verwenden */
sleep_enable(); /* Sleep-Modus aktivieren */
sleep_cpu(); /* Sleep-Modus betreten */
sleep_disable(); /* Empfohlen: Sleep-Modus danach deaktivieren */
```


2 Passives Warten auf Unterbrechungen

- Polling bei passivem Warten nicht möglich (warum?)
- Beispiel:

```
volatile static char event=0;
ISR (INT2_vect) { event=1; }

void main(void) {
   while(1) {

      while(event==0) { /* Warte auf Event */
         sleep_enable();

         sleep_cpu();
         sleep_disable();

      }

      /* bearbeite Event */
```

- Synchronisation erforderlich?

2 Passives Warten auf Unterbrechungen

- Was passiert, wenn der Interrupt wie unten gezeigt eintrifft?
- Beispiel:

```
volatile static char event=0;
ISR (INT2_vect) { event=1; }

void main(void) {
   while(1) {

      while(event==0) { /* Warte auf Event */
         sleep_enable();
                          Interrupt!
         sleep_cpu();
         sleep_disable();

      }

      /* bearbeite Event */
```


2 Dornröschenschlaf vermeiden

- Atomarität von Wartebedingungsprüfung und Sleep-Modus-Aktivierung
- Beispiel:

```
volatile static char event=0;
ISR (INT2_vect) { event=1; }

void main(void) {
   while(1) {
      cli();
      while(event==0) { /* Warte auf Event */
         sleep_enable();
         sei();
         sleep_cpu();          kritischer Abschnitt
         sleep_disable();
         cli();
      }
      sei();
      /* bearbeite Event */
```

- `sei` und die Folgeanweisung werden atomar ausgeführt (notwendig?)

U3-3 Aufgabe 4: Ampelsteuerung

- Fußgängerüberweg mit zwei Ampeln
  - Auto-Ampel: Rot (RED0), Gelb (YELLOW0) und Grün (GREEN0)
  - Fußgängerampel: Rot (RED1), Grün (GREEN1), Signal kommt (BLUE1)
- Autoampel immer min. 10 Sek. grün (Restzeitdarstellung auf 7-Segment)
- Fußgängerampel immer 5 Sek. grün (Restzeitdarstellung auf 7-Segment)
- Umschaltungsanforderung durch Betätigung von Taster BUTTON0
  - Aktivierung der Signal kommt LED
  - Umschaltung, wenn die min. Grünphase der Auto-Ampel verstrichen ist
  - 2 Sek. Warten, nachdem Auto-Ampel rot, bevor FG-Ampel auf grün schaltet
  - Taster wird nach wieder aktiv, nachdem die FG-Ampel auf rot geschaltet hat
- Zwischenzustände während der Umschaltung sind je 500ms aktiv (Auto-Ampel gelb bzw. gelb-rot)