Microcontroller Teil 5 Interrupts

DI (FH) Andreas Pötscher

HTL Litec

Polling vs Interrupts

Polling nennt man das zyklische Abfragen eines „Hardware-Zustandes“ (z. B. des Zustands eines GPIO-Eingangs). Der aktuelle Zustand ist dabei nicht interessant, sondern es wird darauf gewartet, ob sich der Zustand geändert hat.

Polling Beispiel

- Ein Sensor an der rotierenden Achse eines Motors liefert bei jeder Umdrehung einen High-Impuls. Der Motor dreht sich mit bis zu 6.000 Umdrehungen pro Minute.

Polling Beispiel

- Ein Sensor an der rotierenden Achse eines Motors liefert bei jeder Umdrehung einen High-Impuls. Der Motor dreht sich mit bis zu 6.000 Umdrehungen pro Minute.
- Das Sensorsignal ist an einen GPIO-Pin eines Mikrocontrollers angeschlossen, sodass dieser die Anzahl der bisherigen Umdrehungen des Motors erfassen kann.

Polling Beispiel

- Ein Sensor an der rotierenden Achse eines Motors liefert bei jeder Umdrehung einen High-Impuls. Der Motor dreht sich mit bis zu 6.000 Umdrehungen pro Minute.
- Das Sensorsignal ist an einen GPIO-Pin eines Mikrocontrollers angeschlossen, sodass dieser die Anzahl der bisherigen Umdrehungen des Motors erfassen kann.
- Das Signal am GPIO-Pin ist für eine halbe Motorumdrehung High und für die restliche halbe Umdrehung Low. Wie oft muss der Mikrocontroller den Zustand des GPIO-Pins abfragen?

Polling Beispiel

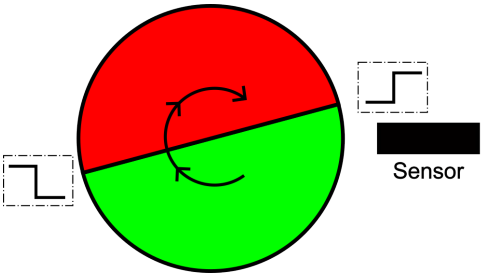

Figure 1: Schema des Beispiels

Lösung

- Motor-Drehzahl:

$$n = 6.000 \frac{\text{Umdrehungen}}{\text{Minute}} = 100 \frac{\text{Umdrehungen}}{\text{Sekunde}}$$

Lösung

- Motor-Drehzahl:

$$n = 6.000 \frac{\text{Umdrehungen}}{\text{Minute}} = 100 \frac{\text{Umdrehungen}}{\text{Sekunde}}$$

- Dauer einer Umdrehung:

$$T = \frac{1}{100\,\text{s}^{-1}} = 10\,\text{ms}$$

Lösung

- Motor-Drehzahl:

$$n = 6.000 \frac{\text{Umdrehungen}}{\text{Minute}} = 100 \frac{\text{Umdrehungen}}{\text{Sekunde}}$$

- Dauer einer Umdrehung:

$$T = \frac{1}{100\,\text{s}^{-1}} = 10\,\text{ms}$$

- Da das Signal für eine halbe Umdrehung High und für die andere Hälfte Low ist, wechselt es seinen Zustand nach:

$$\frac{T}{2} = 5\,\text{ms}$$

Lösung

- Der Mikrocontroller muss daher mindestens alle **5 Millisekunden** den Zustand des GPIO-Pins abfragen („pollen“), da sonst ein gesamter High-Impuls und damit eine komplette Motorumdrehung verloren gehen könnte.

Lösung

- Der Mikrocontroller muss daher mindestens alle **5 Millisekunden** den Zustand des GPIO-Pins abfragen („pollen“), da sonst ein gesamter High-Impuls und damit eine komplette Motorumdrehung verloren gehen könnte.
- Polling ist sehr rechenintensiv, vor allem, wenn Zustände in kurzen Zeitabständen abgefragt werden müssen.

Lösung

- Der Mikrocontroller muss daher mindestens alle **5 Millisekunden** den Zustand des GPIO-Pins abfragen („pollen“), da sonst ein gesamter High-Impuls und damit eine komplette Motorumdrehung verloren gehen könnte.
- Polling ist sehr rechenintensiv, vor allem, wenn Zustände in kurzen Zeitabständen abgefragt werden müssen.
- Wenn der Mikrocontroller in jedem Programmzyklus neben dem Polling noch eine weitere Aufgabe ausführen muss, darf diese maximal so lange dauern, wie es die Zykluszeit des Pollings erlaubt. Ein Durchlauf der Endlosschleife im Hauptprogramm des Mikrocontrollers darf also höchstens so lang sein wie die Polling-Zykluszeit.

Lösung

- Der Mikrocontroller muss daher mindestens alle **5 Millisekunden** den Zustand des GPIO-Pins abfragen („pollen“), da sonst ein gesamter High-Impuls und damit eine komplette Motorumdrehung verloren gehen könnte.
- Polling ist sehr rechenintensiv, vor allem, wenn Zustände in kurzen Zeitabständen abgefragt werden müssen.
- Wenn der Mikrocontroller in jedem Programmzyklus neben dem Polling noch eine weitere Aufgabe ausführen muss, darf diese maximal so lange dauern, wie es die Zykluszeit des Pollings erlaubt. Ein Durchlauf der Endlosschleife im Hauptprogramm des Mikrocontrollers darf also höchstens so lang sein wie die Polling-Zykluszeit.
- Länger andauernde Aufgaben (z. B. die Ausgabe eines Strings über die serielle Schnittstelle) wären dann nicht mehr möglich.

Interrupts

- Aus diesem Grund verfügen Prozessoren und Mikrocontroller über sogenannte **Interrupts**. Dabei erkennt die Peripherie unabhängig von der CPU und dem laufenden Programm, dass sich ein Hardware-Zustand verändert hat.

Interrupts

- Aus diesem Grund verfügen Prozessoren und Mikrocontroller über sogenannte **Interrupts**. Dabei erkennt die Peripherie unabhängig von der CPU und dem laufenden Programm, dass sich ein Hardware-Zustand verändert hat.
- Die Peripherie signalisiert der CPU, dass ein solches Ereignis aufgetreten ist, indem sie ein sogenanntes **Interrupt-Flag** (ein Bit in einem Special-Function-Register) auf eins setzt. Sobald die CPU erkennt, dass ein Interrupt-Flag gesetzt wurde, unterbricht sie das aktuelle Programm und führt das zugehörige Unterprogramm aus.

Interrupts

- Aus diesem Grund verfügen Prozessoren und Mikrocontroller über sogenannte **Interrupts**. Dabei erkennt die Peripherie unabhängig von der CPU und dem laufenden Programm, dass sich ein Hardware-Zustand verändert hat.
- Die Peripherie signalisiert der CPU, dass ein solches Ereignis aufgetreten ist, indem sie ein sogenanntes **Interrupt-Flag** (ein Bit in einem Special-Function-Register) auf eins setzt. Sobald die CPU erkennt, dass ein Interrupt-Flag gesetzt wurde, unterbricht sie das aktuelle Programm und führt das zugehörige Unterprogramm aus.
- Ein solches Unterprogramm wird als **Interrupt-Service-Routine (ISR)** oder **Interrupt-Handler** bezeichnet. Nachdem die Interrupt-Service-Routine vollständig abgearbeitet wurde, setzt die CPU das Hauptprogramm genau an der Stelle fort, an der es zuvor unterbrochen wurde.

Interrupts

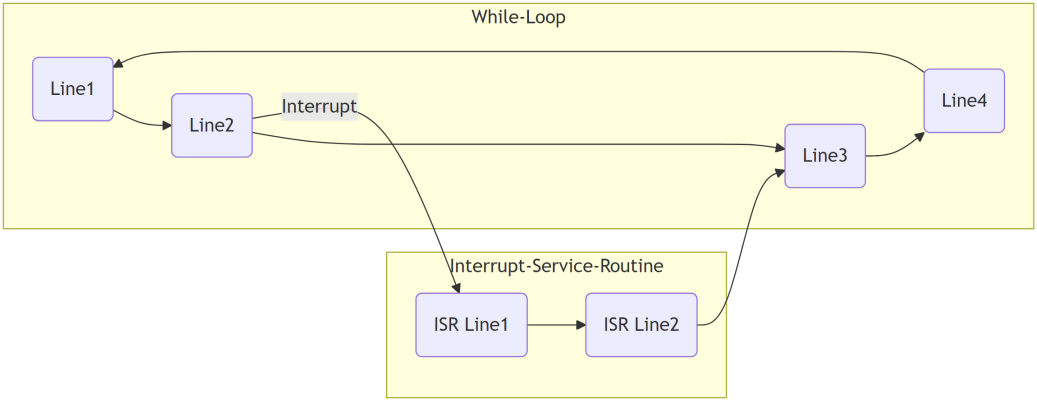

Figure 2: Ablauf eines Interrupt mit zugehöriger Interrupt Service Routine. Nach Line2 kann ein Interrupt auftreten.

Lösung mit Interrupts

- Der Sensorausgang muss mit einem interrupt-fähigen GPIO-Eingang des Mikrocontrollers verbunden werden. In der Setup-Phase des Programms muss der Interrupt aktiviert („enabled“) werden.

Lösung mit Interrupts

- Der Sensorausgang muss mit einem interrupt-fähigen GPIO-Eingang des Mikrocontrollers verbunden werden. In der Setup-Phase des Programms muss der Interrupt aktiviert („enabled“) werden.
- Der Interrupt soll bei steigenden Flanken am GPIO-Eingang (also bei einem Pegelwechsel von Low nach High) ausgelöst werden. In der Interrupt-Service-Routine (ISR) wird eine globale Variable inkrementiert, die die Anzahl der bisher vom Motor ausgeführten Umdrehungen speichert.

Lösung mit Interrupts

- Der Sensorausgang muss mit einem interrupt-fähigen GPIO-Eingang des Mikrocontrollers verbunden werden. In der Setup-Phase des Programms muss der Interrupt aktiviert („enabled“) werden.
- Der Interrupt soll bei steigenden Flanken am GPIO-Eingang (also bei einem Pegelwechsel von Low nach High) ausgelöst werden. In der Interrupt-Service-Routine (ISR) wird eine globale Variable inkrementiert, die die Anzahl der bisher vom Motor ausgeführten Umdrehungen speichert.
- Im Hauptprogramm kann dieser Wert jederzeit ausgelesen werden. Beispielsweise könnte der Motor gestoppt werden, sobald 1.000 Umdrehungen (Windungen) erreicht sind.

Interrupt Flags

- Jedes Mal, wenn die Peripherie des Mikrocontrollers ein Interrupt-Ereignis erkennt, wird das zugehörige Interrupt-Flag gesetzt.

Interrupt Flags

- Jedes Mal, wenn die Peripherie des Mikrocontrollers ein Interrupt-Ereignis erkennt, wird das zugehörige Interrupt-Flag gesetzt.
- Der ATmega2560 verfügt über etwa 56 verschiedene Hardware-Ereignisse, die einen Interrupt auslösen können. Entsprechend gibt es 56 Interrupt-Flags und 56 Interrupt-Service-Routinen (ISRs).

Interrupt Flags

- Jedes Mal, wenn die Peripherie des Mikrocontrollers ein Interrupt-Ereignis erkennt, wird das zugehörige Interrupt-Flag gesetzt.
- Der ATmega2560 verfügt über etwa 56 verschiedene Hardware-Ereignisse, die einen Interrupt auslösen können. Entsprechend gibt es 56 Interrupt-Flags und 56 Interrupt-Service-Routinen (ISRs).
- Ein Interrupt wird aber nur ausgelöst, wenn der zugehörige Interrupt muss aktiviert („enabled“) sein und

Interrupt Flags

- Jedes Mal, wenn die Peripherie des Mikrocontrollers ein Interrupt-Ereignis erkennt, wird das zugehörige Interrupt-Flag gesetzt.
- Der ATmega2560 verfügt über etwa 56 verschiedene Hardware-Ereignisse, die einen Interrupt auslösen können. Entsprechend gibt es 56 Interrupt-Flags und 56 Interrupt-Service-Routinen (ISRs).
- Ein Interrupt wird aber nur ausgelöst, wenn der zugehörige Interrupt muss aktiviert („enabled“) sein und
- Interrupts global erlaubt sind.

Interrupt Vektor

- Ist ein Interrupt enabled unterbricht das Setzen des Interrupt-Flags durch die Peripherie das Hauptprogramm. Die CPU springt dann zu einer festen Adresse im Programmspeicher.

Interrupt Vektor

- Ist ein Interrupt enabled unterbricht das Setzen des Interrupt-Flags durch die Peripherie das Hauptprogramm. Die CPU springt dann zu einer festen Adresse im Programmspeicher.
- Für die 56 Interrupt-Ereignisse im ATmega2560 existieren entsprechend 56 feste Adressen am Beginn des Programmspeichers. Dieser Bereich wird als Interrupt-Vektor-Tabelle bezeichnet, und die 56 festen Adressen nennt man Interrupt-Vektoren.

Interrupt Vektor

- Ist ein Interrupt enabled unterbricht das Setzen des Interrupt-Flags durch die Peripherie das Hauptprogramm. Die CPU springt dann zu einer festen Adresse im Programmspeicher.
- Für die 56 Interrupt-Ereignisse im ATmega2560 existieren entsprechend 56 feste Adressen am Beginn des Programmspeichers. Dieser Bereich wird als Interrupt-Vektor-Tabelle bezeichnet, und die 56 festen Adressen nennt man Interrupt-Vektoren.
- An jeder dieser festen Adressen befindet sich ein Sprungbefehl, der zur eigentlichen, vom Programmierer erstellten Interrupt-Service-Routine (ISR) führt.

Beispiel Interruptvektortabelle

Beispiele aus der Interruptvektortabelle des ATmega 2560 sind:

Vektornummer	Quelle	Beschreibung
2	Int0	Pegeländerung an Pin PD0/INT0
26	USART0 RX	Serielle Schnittstelle Nr. 0: Zeichen wurde empfangen
...	.....	.....

Genauer Ablauf eines Interrupts

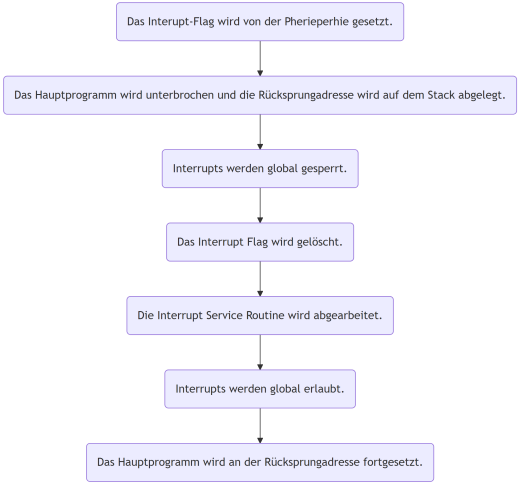

Figure 3: Genauer Ablauf eines Interrupts

Gleichzeite Interrupts

- Wenn während der Abarbeitung einer Interrupt-Service-Routine (ISR) ein weiteres Interrupt-Ereignis auftritt, geht dieses nicht verloren, da die Peripherie das entsprechende Interrupt-Flag setzt (dieses wurde zuvor von der CPU beim Sprung zur ISR gelöscht).

Gleichzeite Interrupts

- Wenn während der Abarbeitung einer Interrupt-Service-Routine (ISR) ein weiteres Interrupt-Ereignis auftritt, geht dieses nicht verloren, da die Peripherie das entsprechende Interrupt-Flag setzt (dieses wurde zuvor von der CPU beim Sprung zur ISR gelöscht).
- Allerdings bleiben Interrupts während der aktuellen Abarbeitung der ISR global gesperrt. Sobald die ISR vollständig abgearbeitet ist, springt die CPU zurück ins Hauptprogramm und aktiviert die Interrupts wieder global. Danach wird zunächst ein einzelner Maschinenbefehl des Hauptprogramms ausgeführt, bevor die CPU das gesetzte Interrupt-Flag erkennt und erneut in die ISR verzweigt.

Gleichzeite Interrupts

- Wenn während der Abarbeitung einer Interrupt-Service-Routine (ISR) ein weiteres Interrupt-Ereignis auftritt, geht dieses nicht verloren, da die Peripherie das entsprechende Interrupt-Flag setzt (dieses wurde zuvor von der CPU beim Sprung zur ISR gelöscht).
- Allerdings bleiben Interrupts während der aktuellen Abarbeitung der ISR global gesperrt. Sobald die ISR vollständig abgearbeitet ist, springt die CPU zurück ins Hauptprogramm und aktiviert die Interrupts wieder global. Danach wird zunächst ein einzelner Maschinenbefehl des Hauptprogramms ausgeführt, bevor die CPU das gesetzte Interrupt-Flag erkennt und erneut in die ISR verzweigt.
- Danach erkennt die CPU, dass das Interrupt-Flag erneut gesetzt ist, und verzweigt direkt wieder in die Interrupt-Service-Routine. Das Interrupt-Ereignis geht also nicht verloren, aber es wird verspätet abgearbeitet.

- Wenn mehrere Interrupts gleichzeitig auftreten, wird der Interrupt mit der höchsten Priorität (niedrigste Vektor-Adresse) zuerst ausgeführt.

- Wenn mehrere Interrupts gleichzeitig auftreten, wird der Interrupt mit der höchsten Priorität (niedrigste Vektor-Adresse) zuerst ausgeführt.
- Falls ein Interrupt-Flag erneut gesetzt wird, während die ISR für denselben Interrupt noch läuft, wird das Ereignis nicht mehrfach gezählt, sondern geht verloren.

Externe Interrupts beim ATmega 2560

- Externe Interrupts werden durch eine Änderung des Spannungspegels an speziellen Interrupt-Pins ausgelöst.

Externe Interrupts beim ATmega 2560

- Externe Interrupts werden durch eine Änderung des Spannungspegels an speziellen Interrupt-Pins ausgelöst.
- Im Gegensatz dazu entstehen interne Interrupts durch Zustandsänderungen in den integrierten Peripherie-Modulen des Mikrocontrollers.

Externe Interrupts beim ATmega 2560

- Externe Interrupts werden durch eine Änderung des Spannungspegels an speziellen Interrupt-Pins ausgelöst.
- Im Gegensatz dazu entstehen interne Interrupts durch Zustandsänderungen in den integrierten Peripherie-Modulen des Mikrocontrollers.
- Der ATmega2560 verfügt über acht externe Interrupts, die auf den folgenden Pins liegen:

Externe Interrupts beim ATmega 2560

- Externe Interrupts werden durch eine Änderung des Spannungspegels an speziellen Interrupt-Pins ausgelöst.
- Im Gegensatz dazu entstehen interne Interrupts durch Zustandsänderungen in den integrierten Peripherie-Modulen des Mikrocontrollers.
- Der ATmega2560 verfügt über acht externe Interrupts, die auf den folgenden Pins liegen:
- INT0 bis INT3 $\rightarrow$ Port D (Pins PD0 bis PD3)

Externe Interrupts beim ATmega 2560

- Externe Interrupts werden durch eine Änderung des Spannungspegels an speziellen Interrupt-Pins ausgelöst.
- Im Gegensatz dazu entstehen interne Interrupts durch Zustandsänderungen in den integrierten Peripherie-Modulen des Mikrocontrollers.
- Der ATmega2560 verfügt über acht externe Interrupts, die auf den folgenden Pins liegen:
- INT0 bis INT3 $\rightarrow$ Port D (Pins PD0 bis PD3)
- INT4 bis INT7 $\rightarrow$ Port E (Pins PE4 bis PE7)

Initialisieren der Interrupt-Pins INT0 bis INT7

Für jeden Interrupt-Pin gibt es vier Möglichkeiten, wie der Interrupt ausgelöst werden kann:

- Ein Low-Pegel löst einen Interrupt aus (Pegel-getriggerter Interrupt, auch level-triggered interrupt genannt).

Initialisieren der Interrupt-Pins INT0 bis INT7

Für jeden Interrupt-Pin gibt es vier Möglichkeiten, wie der Interrupt ausgelöst werden kann:

- Ein Low-Pegel löst einen Interrupt aus (Pegel-getriggerter Interrupt, auch level-triggered interrupt genannt).
- Eine fallende Flanke am Pin löst einen Interrupt aus (Pegeländerung von High nach Low).

Initialisieren der Interrupt-Pins INT0 bis INT7

Für jeden Interrupt-Pin gibt es vier Möglichkeiten, wie der Interrupt ausgelöst werden kann:

- Ein Low-Pegel löst einen Interrupt aus (Pegel-getriggerter Interrupt, auch level-triggered interrupt genannt).
- Eine fallende Flanke am Pin löst einen Interrupt aus (Pegeländerung von High nach Low).
- Eine steigende Flanke am Pin löst einen Interrupt aus (Pegeländerung von Low nach High).

Initialisieren der Interrupt-Pins INT0 bis INT7

Für jeden Interrupt-Pin gibt es vier Möglichkeiten, wie der Interrupt ausgelöst werden kann:

- Ein Low-Pegel löst einen Interrupt aus (Pegel-getriggerter Interrupt, auch level-triggered interrupt genannt).
- Eine fallende Flanke am Pin löst einen Interrupt aus (Pegeländerung von High nach Low).
- Eine steigende Flanke am Pin löst einen Interrupt aus (Pegeländerung von Low nach High).
- Sowohl eine steigende als auch eine fallende Flanke am Pin lösen einen Interrupt aus.

Initialisieren der Interrupt-Pins INT0 bis INT7

Jeder Interrupt-Pin hat ein zwei Bit breites Bitmuster, das aus den Bits ISCn1 und ISCn0 besteht. Das „n“ steht dabei für die Interrupt-Nummer (0 bis 7). Diese acht mal zwei Bits verteilen sich auf die beiden Register EICRA (External Interrupt Control Register A) und EICRB (External Interrupt Control Register B).

Bit	7	6	5	4	3	2	1	0
EICRA:	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00

Bit	7	6	5	4	3	2	1	0
EICRB:	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40

Initialisieren der Interrupt-Pins INT0 bis INT7

- ISC steht für Interrupt Sense Control.

Initialisieren der Interrupt-Pins INT0 bis INT7

- ISC steht für Interrupt Sense Control.
- Die Bits ISC71 und ISC70 gehören zum externen Interrupt des Pins PE7/INT7, die Bits ISC61 und ISC60 zum Pin PE6/INT6, und so weiter.

Initialisieren der Interrupt-Pins INT0 bis INT7

- ISC steht für Interrupt Sense Control.
- Die Bits ISC71 und ISC70 gehören zum externen Interrupt des Pins PE7/INT7, die Bits ISC61 und ISC60 zum Pin PE6/INT6, und so weiter.
- Übrigens spricht man bei dem Bit ISC61 vom Bit „ISC sechs eins“ und nicht vom Bit „ISC einundsechzig“.

Initialisieren der Interrupt-Pins INT0 bis INT7

Die Bits haben folgende Bedeutung:

Bit ISCn1	**Bit ISCn0**	**Bedeutung:**
0	0	Low-Level: Der Low-Pegel am Pin INTn erzeugt den Interrupt (level-triggered Interrupt).
0	1	Any Edge: Sowohl eine steigende als auch eine fallende Flanke am Pin INTn erzeugen einen Interrupt.
1	0	Falling edge: Die fallende Flanke am Pin INTn erzeugt einen Interrupt
1	1	Rising edge: Die steigende Flanke am Pin INTn erzeugt einen Interrupt.

Beispiel

Beispiel:
Der Pin PE5/INT5 soll bei fallenden Flanken Interrupt-Ereignisse auslösen. Zu Testzwecken werden die Interrupts durch einen Taster ausgelöst, der zwischen dem Pin und GND geschaltet wird. Welche Schritte sind dafür notwendig?

Beispiel

LINZER TECHNIKUM
HTL Paul-Hahn-Straße

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

Pin initialsieren. PE5/INT5 wird ein Eingang und der interne Pull-Up wird aktiviert.

```
DDRE &= ~(0x01 << 5);
PORTE |= 0x01 << 5;
```

Beispiel

Fallende Flanken sollen Interrupts auslösen: Die beiden Bits ISC51 und ISC50 müssen auf das Bitmuster 10 gesetzt werden. Beide Bits sind im Register `EICRB` an den Bitpositionen 3 und 2. Für diese können die Macros ISC51 (`#define 3`) und ISC50 (`#define 2`) verwendet werden.

```
EICRB |= 0x01 << ISC51;
EICRB &= ~(0x01 << ISC50);
```

Register für Interrupt-Flags

- Die Interrupt-Flags für diese acht externen Interrupts befinden sich im Special-Function-Register EIFR.
- Diese werden aber nur selten direkt verwendet. Sondern automatisch von der Hardware gesetzt und wieder gelöscht.

Bit	7	6	5	4	3	2	1	0
EIFR:	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	INTF0

Register für Interrupt-Enable-Flags

Die Interrupt-Enable-Flags für die acht externen Interrupts befinden sich im Special-Function-Register EIMSK.

Bit	7	6	5	4	3	2	1	0
EIMSK:	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0

Beispiel

Beispiel:
Spannungspegeländerungen am Pin PE5/INT5 sollen Interrupts auslösen können, während Änderungen an den anderen sieben Interrupt-Pins keine Interrupts auslösen dürfen. Mit welchem Wert muss das EIMSK-Register beschrieben werden?

Beispiel

Das Bit „INT5“, das ist das Bit Nr. 5 im EIMSK-Register muss auf eins gesetzt werden, die anderen auf null. Der Befehl lautet:

```
EIMSK = (0x01 << INT5);
```

Globales Erlauben/Sperren von Interrupts

- Jeder Prozessor und jeder Mikrocontroller verfügt über eine einfache Möglichkeit, Interrupts global zu sperren oder zu erlauben.

Globales Erlauben/Sperren von Interrupts

- Jeder Prozessor und jeder Mikrocontroller verfügt über eine einfache Möglichkeit, Interrupts global zu sperren oder zu erlauben.
- Beim ATmega2560 werden Interrupts global aktiviert, indem das sogenannte „I“-Bit auf 1 gesetzt wird. Sind Interrupts global gesperrt, ist dieses Bit auf 0 gesetzt.

Globales Erlauben/Sperren von Interrupts

- Jeder Prozessor und jeder Mikrocontroller verfügt über eine einfache Möglichkeit, Interrupts global zu sperren oder zu erlauben.
- Beim ATmega2560 werden Interrupts global aktiviert, indem das sogenannte „I“-Bit auf 1 gesetzt wird. Sind Interrupts global gesperrt, ist dieses Bit auf 0 gesetzt.
- In der Programmiersprache C stehen dafür folgende Funktionen zur Verfügung:

Globales Erlauben/Sperren von Interrupts

- Jeder Prozessor und jeder Mikrocontroller verfügt über eine einfache Möglichkeit, Interrupts global zu sperren oder zu erlauben.
- Beim ATmega2560 werden Interrupts global aktiviert, indem das sogenannte „I“-Bit auf 1 gesetzt wird. Sind Interrupts global gesperrt, ist dieses Bit auf 0 gesetzt.
- In der Programmiersprache C stehen dafür folgende Funktionen zur Verfügung:
- `sei();` zum globalen Erlauben (enable) von Interrupts.

Globales Erlauben/Sperren von Interrupts

- Jeder Prozessor und jeder Mikrocontroller verfügt über eine einfache Möglichkeit, Interrupts global zu sperren oder zu erlauben.
- Beim ATmega2560 werden Interrupts global aktiviert, indem das sogenannte „I“-Bit auf 1 gesetzt wird. Sind Interrupts global gesperrt, ist dieses Bit auf 0 gesetzt.
- In der Programmiersprache C stehen dafür folgende Funktionen zur Verfügung:
- `sei();` zum globalen Erlauben (enable) von Interrupts.
- `cli();` zum globalen Sperren (disable) von Interrupts.

Globales Erlauben/Sperren von Interrupts

- Wenn Interrupts mit `cli();` global gesperrt werden, sind alle Interrupts deaktiviert. Unabhängig von den einzelnen Interrupt-Enable-Flags.

Globales Erlauben/Sperren von Interrupts

LINZER TECHNIKUM
HTL Paul-Hahn-Straße

- Wenn Interrupts mit `cli();` global gesperrt werden, sind alle Interrupts deaktiviert. Unabhängig von den einzelnen Interrupt-Enable-Flags.
- Wenn Interrupts mit `sei();` global erlaubt werden, sind nur jene Interrupts aktiv, deren zugehöriges Interrupt-Enable-Flag ebenfalls auf 1 gesetzt ist.

Das ISR() Makro

- Wie bei regulären Funktionen wird die ISR außerhalb anderer Funktionen definiert und darf nicht innerhalb der `main()`-Funktion verschachtelt werden.

Das ISR() Makro

- Wie bei regulären Funktionen wird die ISR außerhalb anderer Funktionen definiert und darf nicht innerhalb der `main()`-Funktion verschachtelt werden.
- Der Funktionskopf wird mit dem ISR()-Makro gebildet.

Das ISR() Makro

- Wie bei regulären Funktionen wird die ISR außerhalb anderer Funktionen definiert und darf nicht innerhalb der `main()`-Funktion verschachtelt werden.
- Der Funktionskopf wird mit dem ISR()-Makro gebildet.
- Innerhalb der runden Klammern des ISR-Makros muss der korrekte Interrupt-Vektor angegeben werden. Da es insgesamt 56 Interrupt-Ereignisse gibt, muss dem Compiler mitgeteilt werden, zu welchem Ereignis die ISR gehört.

Das ISR() Makro

Die folgende Tabelle listet die Namen einiger Interrupt-Vektoren, wie sie in der Programmiersprache C für das ISR-Makro verwendet werden müssen. Die Namen aller Interrupt-Vektoren enden mit _vect:

Interrupt-Vektor	**Interrupt-Quelle**
`INT0_vect`	Externer Interrupt 0 (Pin PD0/INT0)
`INT1_vect`	Externer Interrupt 1 (Pin PD1/INT1)
...	...
`TIMER1_COMPA_vect`	Timer1, Comparator A
`TIMER1_OVF_vect`	Timer1, Overflow
...	...

Das ISR() Makro

LINZER TECHNIKUM
HTL Paul-Hahn-Straße

Ein konkreter Aufruf des ISR()-Makros sieht so aus:

```
int main()
{
    //main
}

ISR(INT2_vect)
{
    //Interrupt Service Routine
}
```

Datenaustausch zwischen main und der ISR

LINZER TECHNIKUM
HTL Paul-Hahn-Straße

In den meisten Fällen müssen zwischen dem Hauptprogramm und der Interrupt-Service-Routine Daten ausgetauscht werden. Dazu werden globale Variablen genutzt.

```
volatile int counter = 0;

int main()
{
    printf("%d\n", counter);
}

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

Datenaustausch zwischen main und der ISR

- Der Typ-Qualifizierer `volatile` weist den Compiler an, dass sich der Wert einer Variable jederzeit unerwartet ändern kann.

Datenaustausch zwischen main und der ISR

LINZER TECHNIKUM
HTL Paul-Hahn-Straße

- Der Typ-Qualifizierer `volatile` weist den Compiler an, dass sich der Wert einer Variable jederzeit unerwartet ändern kann.
- Dies ist besonders wichtig für Variablen, die durch Hardware oder Interrupt-Service-Routinen modifiziert werden.

Datenaustausch zwischen main und der ISR

- Der Typ-Qualifizierer `volatile` weist den Compiler an, dass sich der Wert einer Variable jederzeit unerwartet ändern kann.
- Dies ist besonders wichtig für Variablen, die durch Hardware oder Interrupt-Service-Routinen modifiziert werden.
- Wird `volatile` vergessen, kann der Compiler anstelle eines direkten Speicherzugriffs einen zuvor zwischengespeicherten Wert aus einem Register verwenden.

Datenaustausch zwischen main und der ISR

- Der Typ-Qualifizierer `volatile` weist den Compiler an, dass sich der Wert einer Variable jederzeit unerwartet ändern kann.
- Dies ist besonders wichtig für Variablen, die durch Hardware oder Interrupt-Service-Routinen modifiziert werden.
- Wird `volatile` vergessen, kann der Compiler anstelle eines direkten Speicherzugriffs einen zuvor zwischengespeicherten Wert aus einem Register verwenden.
- In solchen Fällen bemerkt das Hauptprogramm nicht, dass die Variable durch eine Interrupt-Service-Routine (ISR) verändert wurde.

Gesamter Programmaufbau

Bei der Programmierung einer Mikrocontroller-Anwendung mit Interrupts sollten folgende Punkte beachtet werden:

- Kurze ISR-Ausführung

Gesamter Programmaufbau

Bei der Programmierung einer Mikrocontroller-Anwendung mit Interrupts sollten folgende Punkte beachtet werden:

- Kurze ISR-Ausführung
- Datenaustausch über globale Variablen

Gesamter Programmaufbau

Bei der Programmierung einer Mikrocontroller-Anwendung mit Interrupts sollten folgende Punkte beachtet werden:

- Kurze ISR-Ausführung
- Datenaustausch über globale Variablen
- volatile für ISR-Variablen

Gesamter Programmaufbau

Bei der Programmierung einer Mikrocontroller-Anwendung mit Interrupts sollten folgende Punkte beachtet werden:

- Kurze ISR-Ausführung
- Datenaustausch über globale Variablen
- volatile für ISR-Variablen
- Aufwendige Berechnungen ins Hauptprogramm auslagern

Beispiel

Beispiel:
An Pin 2 von Port D (PD2/INT2) ist ein Taster gegen Masse geschaltet. Dieser Pin wird als Eingang mit aktiviertem Pull-up-Widerstand konfiguriert. Der externe Interrupt 2 (INT2) wird so eingestellt, dass eine steigende Flanke (das Loslassen des Tasters) einen Interrupt auslöst.
Im Hauptprogramm wird in einer Endlosschleife der aktuelle Wert von `counter` *über die serielle Schnittstelle mittels* `printf()` *ausgegeben.*

Link zum Beispiel