Interrupts

Bei den Arduino Mikrocontrollern sind Interrupts im Grunde ein Signal, das es ermöglicht, eine Funktion in einem Sketch **jederzeit** direkt aufzurufen, gleichgültig, womit der Arduino sonst gerade „beschäftigt“ ist.

Wird ein Interrupt ausgelöst, wird der Ablauf des Programms angehalten, die Interrupt-Funktion ausgeführt [1)], wenn die ISR abgearbeitet ist, wird das das Programm an der Stelle der Unterbrechung fortgeführt.

Wie macht mans?

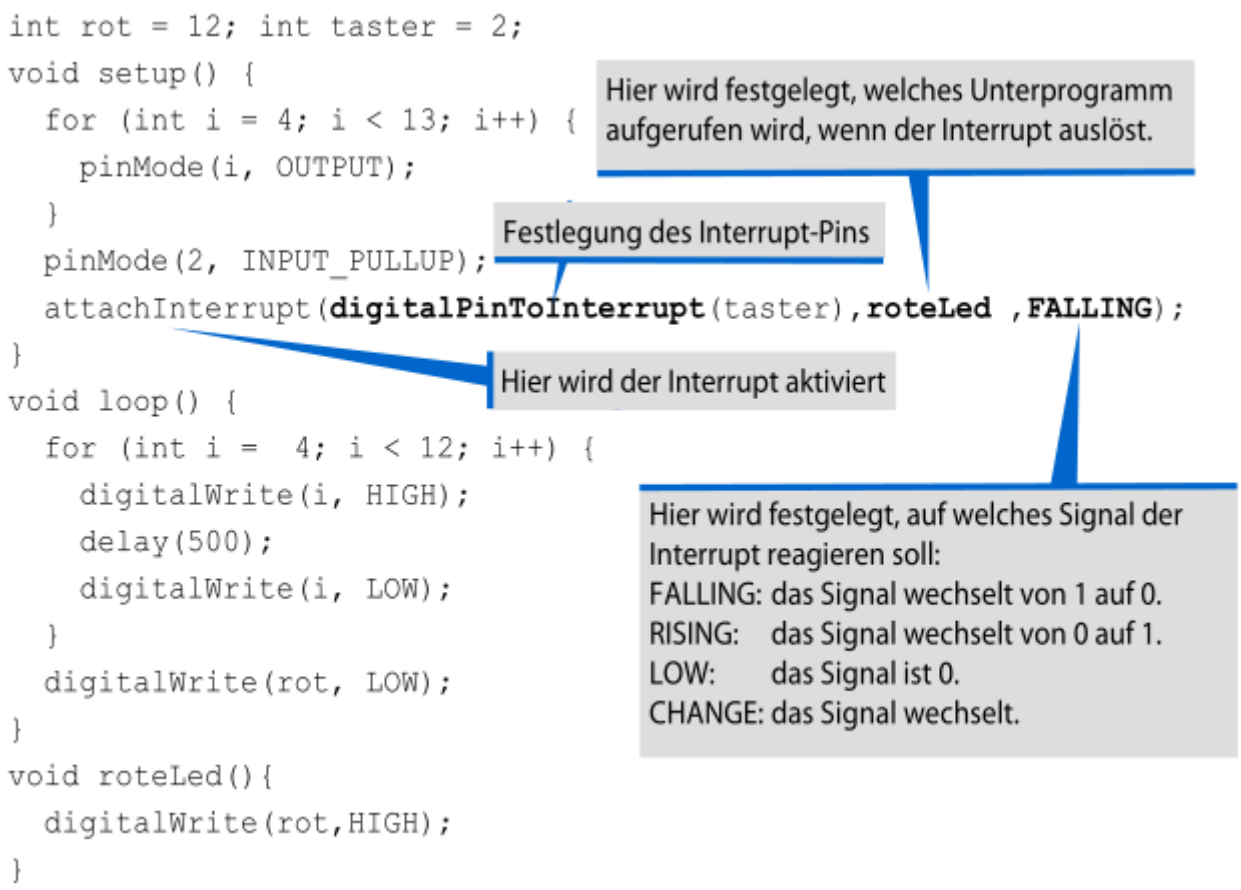
Der Ardunio Uno hat zwei Interrupts, die Pins 2 und 3 können verwendet werden. Um eine Interrupt-Service-Routine mit einem Interrupt zu verknüpfen, verwendet man die folgende Syntax:

```
attachInterrupt(PIN, ISR, mode)
```

dabei ist:

- PIN: Der Pin, der die ISR Triggern soll
- ISR: Der Name der Funktion, die aufgerufen werden soll
- mode: Definiert, wann der Interrupt getriggert werden soll. Dafür sind bereuts 4 Konstanten definiert:
  - LOW Interrupt wird getriggert, wenn der Pin LOW ist
  - CHANGE Interrupt wird getriggert, wenn der Pin den Wert ändert
  - RISING Interrupt wird getriggert, wenn der Pin von LOW auf HIGH wechselt
  - FALLING Interrupt wird getriggert, wenn der Pin von HIGH auf LOW wechselt

Beispiel 1



(A1)

Das Programm steuert ein Lauflicht mit 8 LEDs. Erläutere, wie die LEDs anschlossen sein müssen, damit das funktioniert. Welche Verhaltensweise wird durch den Einsatz des Interrupts erreicht?

ISR-Regeln

Für ISR gelten einige besondere Regeln:

- ISR sollten möglichst kurz gehalten werden - keine serielle Konsole in ISRs!
- ISR können keine Argumente bekommen oder Rückgabewerte zurückgeben. Stattdessen werden - ausnahmsweise! - globale Variablen benutzt, um Daten zwischen Interrupt Service Routinen und dem Hauptprogramm zu tauschen. Damit die Variablen dabei korrekt geändert werden, sollten sie als `volatile` deklariert werden.
- Einige Funktionen verhalten sich in ISRs anders als gewohnt oder funktionieren gar nicht:
  - `millis()` verlässt sich zum Zählen auf Interrupts, wird also in einer Interrupt Service Routine niemals hochzählen.
  - `delay()` benutzt ebenfalls Interrupts und wird deshalb gar nicht in einer Interrupt

Service Routine funktionieren - sollte dort aber aucch n iemals benutzt werden, weil, die ISR ja schnell abgearbeitet werden sollte!
- `micros()` wird anfangs gut funktionieren, sich aber nach etwa 1 bis 2 ms unvorhersehbar verhalten - möglichst nicht benutzen.
- `delayMicroseconds()` benutzt keine Zähler und wird deshalb normal funktionieren.

Beispiel 2

```
int taster = 2;
volatile int zustand = 0;

void setup() {
  for (int i = 4; i < 13; i++) {
    pinMode(i, OUTPUT);
  }
  pinMode(2, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(taster),meineISR ,FALLING);
}
void loop() {
  for (int i =  4; i < 12; i++) {
    digitalWrite(i, HIGH);
    delay(500);
    digitalWrite(i, LOW);

    if(zustand == 1){
       machIrgenwas();
       zustand = 0;  //zurücksetzen der Variablen zustand auf 0
    }
  }
}
void meineISR(){zustand = 1;}

void machIrgenwas(){…} // ist noch leer, ergänze!
```

Variablen im Interrupt müssen als volatile deklariert werden!

(A2)

- Was könnte man mit dem Programmgerüst bei geeigneter ausgestaltung der Funktion `machIrgenwas` erreichen?
- Wie könnte man ein solches Konstrukt einsetzen, um z.B. einen Linienfolger zu steuern oder schnell auf ein Hindernis zu reagieren?

Interrupts aktivieren/deaktivieren

Manchmal möchte man in bestimmten Sequenzen eines Programms eine Unterbrechung des Ablaufs durch Interrupts nicht zulassen, in anderen aber schon. Dazu kann man dem Arduino mitteilen, ob er auf Interrupts reagieren soll oder nicht:

```
noInterrupts(); // Ab jetzt nicht mehr auf IR reagieren!
// anderer Code, der nicht unterbrochen werden soll
interrupts(); // Ab hier wieder auf IR reagieren
```

Weitere Informationen findest du hier: https://gammon.com.au/interrupts

Material

interrupts.odp	1.2 MiB	22.05.2023 19:27
interrupts.pdf	315.5 KiB	22.05.2023 19:27
isr01.png	90.5 KiB	22.05.2023 19:00
isr2.png	62.8 KiB	22.05.2023 19:08

[1)]

„ISR“ - Interrupt Service Routine