RC Receiver

Author: Dipl.-Ing. Wilfried Klaas
Board: Arduino Duemilanove, Arduino Uno, Arduino Leonardo (mit Einschränkungen)

Für die verschiedenen Projekte hier, habe ich mal eine kleine Bibliothek geschrieben, die die Anbindung an einen Modellbauempfänger etwas einfacher macht. Die Bibliothek beinhaltet ein Objekt RCReceive, welches für die Anbindung zuständig ist.
Die Bibliothek ermittelt automatisch den Nullpunkt aus den ersten 10 Werten. Weiterhin wird der Wertebereich automatisch auf den Bereich von 0..255 beschränkt. Neu ist die Fehlererkennung. Werden mehr als 3 fehlerhafte Impulse vom Empfänger erkannt, kann man das in seinem Programm abfragen und entsprechend reagieren, z.B. Failsafe oder RTH.

21.08.2015 Neue Version 1.4.0 in Github
Die Bibliothek ist nun im Library Manager der Arduino IDE zu finden und wird auch auf GITHUB gehosted.
(RCReceiver auf Github)
Die aktuellen Releases findet man ab sofort hier RCReceiver Releases

Die Bibliothek gibt's hier: RCReceiver (letzte Änderung 09.04.2014)
2 Programmtemplates sind mit dabei.

09.04.2014 Neue Version 0.2.0
Jetzt funktioniert die Lib auch endlich mit einem Arduino Mega 256 sauber. Viel Spass damit.

06.10.2013 Neue Version
Ich habe jetzt etliche Tage an einer neuen Version gebastelt. Ziel war es, das ganze zusammen mit einer einfachen Servo Bibliothek auf den ATTiny 25/45/85 zum laufen zu bekommen. Das ist mir auch geglückt. Deswegen gibt es jetzt hier eine neue Version.

31.01.2013 Es gibt eine neue Version.
- Verbesserte Fehlererkennung
- einfachere Initialisierung im Interruptmodus
- Werte nun auch als ms abholbar
- schnellere Interruptroutine durch eigene Timerinitialisierung
- Unterstützung von Arduino Mega und Arduino Leonardo

Benutzt wird dabei der Timer1, der auch von der Servo Bibliothek benutzt wird. Da ich die gleiche Initialisierung mache, ist das aber kein Problem.

Installation

Die Installation ist einfach, Für die Installation einfach das ganze Zip in das Sketchverzeichniss auspacken. Danach sollte dort ein libraries Verzeichniss extistieren und dortdrin sind 2 neue Libs. RCRecive und MCSTools.

Einbinden

Zum Einbinden einfach auf [Sketch]/[Library importieren...], und schon sind in ihrer Quelldatei einfach folgende Zeilen hinzugefügt:

```
#include <RCReceive.h>
```

Das gleiche bitte auch mit der MCStools Bibliothek.

```
#include <debug.h>
#include <makros.h>
#include <RCReceive.h>
```

Für den ATTIny 8 und für den Mega gibt es noch 2 extra #define's.

- **ATTiny8**

über

```
#define USE_TIMER_1
```

kann der Timer 1 anstatt der internen micros() benutzt werden. Dadurch wird die Messung genauer und evt. braucht man den Timer0 noch für was anderes.

- **ATMega256**

über

```
#define USE_TIMER_5
```

kann der Timer 5 anstatt des Timer 1 benutzt werden. Dadurch werden dei PWM Ports auf den Pins 11,12,evt. 13 nicht mehr beeinflußt und evt. braucht man den Timer1 noch für was anderes. Bei Timer 5 gehen dann die PWM auf 44, 45 und 46 (die sind auf dem XIO Stecker) aber nicht mehr.

Die defines kann man entweder direkt in der RCReceive.h aktivieren oder aber man schreibt diese vor dem

```
#include "RCReceive.h"
```

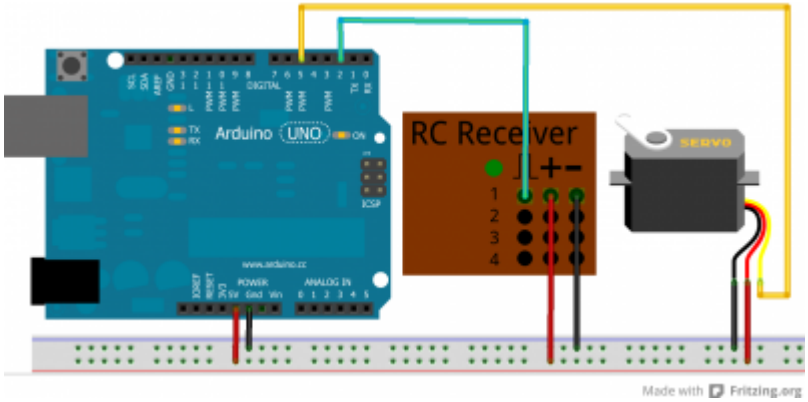
Beispiel:

```
#define USE_TIMER_5
#include <RCReceive.h>
```

Hardwareanbindung

Schliessen Sie einfach die Masse und den Impulspin an den Arduino an. Masse an den entsprechenden Massepin vom Arduino und der Impulspin an den Pin, den Sie im Programm definiert haben.
Beim Testen des Programms ist es sinnvoll, das man den Empfänger vom Arduino mitversorgen kann. Dann, und auch nur dann, können Sie auch den mittleren Pin vom Servokabel an den +5V Pin des Arduino anschliessen. Bitte vermeiden Sie, den Empfänger in dem Fall anderweitig mit Strom zu versorgen. Denken Sie bitte auch daran, daß manche Fahrtregler eine eingebaute Versorgungsfunktion (BEC) haben. Eine Versorgung aus 2 verschiedenen Quellen kann sowohl für den Arduino wie auch für den Empfänger, ESC gefährlich sein. Im Zweifel benutzen Sie dann den Vin Pin bzw. den DC-Anschluss des Arduinos.

Im Pollingbetrieb kann jeder Pin verwendet werden. Beim Interruptbetrieb können nur die Pins 2 und 3 benutzt werden.



Benutzung

Möchten Sie nun einen Kanal auswerten, müssen Sie zunächst einen Pin zur Verfügung stellen. Die korrekte Initialisierung übernimmt die Bibliothek. Dazu müssen Sie pro Kanal eine Variable vom Typ RCRecive definieren und diese mit dem Pin verbinden.

```
...
// Hardwareanbindung für Arduino Hardware, Empfängerkanäle
const byte PIN_RC_STE = 3;
RCReceive rcReceiver;
...
void setup() {
...
  // und Pin mit dem Objekt verbinden
  rcReceiver.attach(PIN_RC_STE);
...
}
```

Soweit so einfach. Es gibt 2 Varianten der Einbindung.

Polling

Beim Polling müssen Sie dem Objekt, denn darum handelt es sich nun, sagen, wann es den Wert vom Empfänger lesen soll. Dieses geschiet mit der Methode `poll()`. Am besten macht man das z.B. am Anfang in der `loop()` Methode.

```
void loop() {
  // neuen RC Wert für Steuerservo lesen.
  rcReceiver.poll();
...
}
```

Die ersten 10 Werte gehen immer in die Nullpunktbestimmung. d.h. wenn man den Arduino einschaltet und dieser vernünftige Werte vom Empfänger erhält, sind die 10 ersten Werte nur für die Nullpunktbestimmung relevant. Das geschiet automatisch im Hintergrund. Es gibt eine Funktion zur Abfrage, ob der Nullpunkt bereits gelesen wurde. `hasNP()`
Den Nullpunkte sollte man also abfragen und solange mit der eigentlichen Funktion warten, bis dieser korrekt bestimmt worden ist.
Machen kann man das z.B. so:

```
...

void loop() {
...
  // Aktuellen RC-Wert lesen
  rcReceiver.poll();

  // Nullpunktsbestimmung ?
  if (rcReceiver.hasNP() && !rcReceiver.hasError()) {
    doWork();
  } else if (rcReceiver.hasError()) {
    // Fehlerbehandlung failsafe oder sowas...
  }
...
}
```

Auch kann man an dieser Stelle nach Fehlern abfragen und entsprechend reagieren.
Um im Programm jetzt den aktuellen Wert herauszulesen, muss man folgende Methode aufrufen:

```
...
  byte value = rcReceiver.getValue();
...
```

Damit erhält man den gemittelten Wert der letzten 10 Werte. Die Mittlung mache ich deswegen, damit etweige Fehler in der Übertragung nicht so ins Gewicht fallen. Natürlich hat das eine Verzögerung zur Folge. Und zwar von genau 200ms.
Nachteil Wenn Sie den Polling Mechanismus einsetzten wollen und zus. noch weitere Bibliotheken, die mit Interrupts arbeiten, kann es sein, das der Wert nicht konstant ist, sondern unruhig wird. Die zur Messung des Empfängerwertes verwendete Funktion `pulseIn()` wird dann von den im System

vorhandenen Interrupts gestört.
Natürlich kann man vor dem `pulseIN()` mit `cli()` die Interrupts abschalten und mit `sei()` wieder einschalten, dann wird der gemessene Wert wieder schön ruhig, aber leider funktionieren dann evt. die anderen Bibliotheken nicht mehr richtig. Sehr Ärgerlich ist, daß gerade die für uns wichtige Servo Bibliothek einer der Störenfriede ist. Und die Servo Bibliothek reagiert ganz schlecht auf das Ausschalten des Interruptes.

Eine Verbesserung bringt da das 2. Messverfahren.

Interrupts

Für die Messung des Empfängersignales kann man auch Interrupts verwenden. Dazu **müssen** aber die Pins 2 und 3 verwendet werden. Andere Pins gehen dann nicht. (Nur Deumillanove und Uno) Nur diese Pins sind mit den Softwareinterrupts 0 und 1 verbinden. D.h. **Pin 2 ist für den Interrupt 0 zuständig und Pin 3 für den Interrupt 1**.
Um den Interrupt zu benutzen müssen wir lediglich zum Initialisieren eine andere Methode verwenden. Also ändert sich der `setup()`-Code zu:

```
void setup() {
  rcReceiver.attachInt(PIN_RC);
  // put your setup code here, to run once:
}
```

Wichtig ist die `attachInt()` Funktion. Der Parameter bezeichnet den richtigen Pin und nicht die Interruptnummer. Die Umsetzung erfolgt intern.
Ein `poll()` in der `loop()` Funktion kann jetzt entfallen.

Hier mal eine Auflistung der verschiedenen möglichen Pins der verschiedenen Boards.

- Duemillanove, Uno (328): Pins 2 und 3
- Leonardo: Pins 3, 2, 0, 1 (Wichtig: 2 und 3 sind vertauscht, macht für uns aber nix, da die Bibliothek alles richtig verdrahtet.)
- Mega2560: Pins 2, 3, 21, 20, 19, 18

Methoden

Hier jetzt mal alle Methoden auf einen Streich.

```
  void attach(byte pin);
```

Verbindet das Objekt mit einem Arduino Pin.

```
  void attachInt(byte pin);
```

Verbindet das Objekt mit einem Arduino Pin und startet den Interruptmodus.

```
  void detachInt(byte pin);
```

zugewiesenen Interruptroutine lösen.

```
byte getValue();
```

Holt den aktuellen gemittelten Wert im Bereich von 0-255.

```
int getMsValue();
```

Holt den aktuellen gemittelten Wert in ms.

```
byte poll();
```

Den aktuellen Wert vom Empfänger holen.

```
byte hasNP();
```

Nullpunkt wurde ermittelt.

```
byte getNP();
```

Aktuellen Nullpunkt holen.

```
byte hasError();
```

Es wurden mehr als 3 fehlerhafte Pakete vom Empfänger übermittelt.