Dokumentation
UniPWM-Library für Arduino® Nano

Inhalt

Versionen

1.00	06/04/2014	Created
1.01	06/28/2014	"Convenience" macros added
1.02	07/09/2014	Optimized ISR. Max. software PWM 10 channels can be serviced now
		New programming interface for simplified usage
		Low battery handling
1.03	01/15/2015	Sleep mode changes - dont let CPU sleep
		Analog pin for voltage measurement can be defined in SetLowBatt(...)
		You can provide the receiver input value in DoLoop(...) to replace the receiver output by some other measurement
1.04	03/04/2015	multiple input channels, new sample4 to demonstrate usage of 2nd rc input
1.05	03/07/2015	storage structure modified to separate workdata from static data (no separate SEQUENCE() for each pin needed anymore)

Download

http://sourceforge.net/projects/arduinounipwmlib/

Lizenzbedingungen

Kurzbeschreibung

Die UniPWM-Library stellt in der Pulsweite variable Signale (PWM=Pulsweitenmodulation) an den Digitalausgängen eines Arduino Nano bereit. Diese Signale sind nicht nur in der Pulsweite einstellbar, sondern auch auf mehreren Kanälen synchron im zeitlichen Verlauf änderbar. Zusätzlich ist ein digitaler Eingangskanal vorhanden, der ein pulsweitenmoduliertes Eingangssignal messen kann.

Die Anwendung ist für den RC-Modellbau vorgesehen.

ACHTUNG: Eine Anwendung in sicherheitsrelevanten Anwendungen (Flugsteuerung) ist NICHT zulässig. Die Software ist nur zur Ansteuerung von Leuchtdioden oder Servos in nicht-sicherheitsrelevanten Anwendungen vorgesehen (z.B. Ein-Ausfahren eines Landescheinwerfers).

Anwendungsfälle:

- LED-Steuerung mit variabler Helligkeit
- Helligkeitsverlauf zeitlich änderbar (z.B. anschwellendes und abschwellendes Blinklicht)
- LED-Steuerung auf mehreren Kanälen synchron (z.B. Positionslichter mit sich abwechselnden ACL-Blitzern)
- Servosteuerung (z.B. für das Ausfahren eines Landescheinwerfers)
- RC-Empfängereingang. Damit lassen sich mehrere unterschiedliche Blinkmuster vom Sender aus umschalten.

Technische Daten

- Läuft auf Arduino Nano (ATMega 328) mit Arduino IDE 1.0x
- Bis zu 14 Software-PWM-Kanäle bei 50 Hz PWM-Frequenz
- 8 bis 10 Software-PWM-Kanäle bei 100 Hz PWM-Frequenz
- Bis zu 6 Software-PWM-Kanäle mit Hardware PWM
- Ansteuerung von LEDs oder digitalen Servos (100Hz, Auflösung ca. 40 Schritte) über Software PWM
- Ansteuerung eines analogen Servo über Hardware PWM-Ausgang an Pin 9 (50 Hz, Auflösung ca. 250 Schritte).
- 1 digitaler Eingang für RC-Empfänger (ca. 3-5 Schaltstellungen detektierbar)
- Periodische oder aperiodische Sequenzen zur Modulation von LED-Helligkeit oder Servostellungen frei programmierbar. Die Ausgabekanäle können zueinander synchronisiert werden (Sequencer).
- Einfache Programmierung über C++ in der Arduino IDE 1.0x.

Weitere Besonderheiten

- Minimale Anzahl externer Bauteile
- Geringe Hardwarekosten
- Komfortable Anschaltung an den PC über USB-Schnittstelle (kein externer ‚Programmer' erforderlich)
- Open-Source-Software

Programmierung

Hardware-Ressourcen:

- Timer 2
- Timer 1, bei Verwendung des analogen Servoausgangs an Pin 9

Grundkonzept

Die Definition der PWM-Verläufe erfolgt über Objekte. Diese werden einem zentralen Steuerobjekt `UniPWMControl` übergeben. Die aktuelle Implementierung geht davon aus, dass die PWM-Objekt einmalig beim Programmstart erzeugt werden.

Dabei ist es möglich, mehrere „Patterns“ zu erzeugen, die durch einen RC-Kanal umgeschaltet werden.

Phasen und Sequenzen

Die Steuerung der Ausgangskanäle erfolgt durch Setzen von PWM-Werten, die für einen bestimmten Zeitraum gültig sind. Der PWM-Wert ist dabei proportional zur Helligkeit einer angeschlossenen LED oder der Stellung eines Servos. Der Verlauf eines PWM-Werts über eine bestimmte Zeit wird *Phase* genannt. Mehrere *Phasen* können aneinandergereiht werden und bilden eine *Sequenz*.

Es gibt zwei Arten von Phasen:

- Konstanter PWM-Wert für eine bestimmte Zeit
- PWM-Wert folgt einer linearen Rampe über eine bestimmte Zeit

Makros zur einfachen Definition von PWM-Phasen und Sequenzen

In UniPWMMacros.h sind eine Reihe von Makros definiert, um Phasen einfach festzulegen:

Gibt einen konstanten Wert über eine bestimmte Zeit aus. Die Dauer ist in allen Befehlen der angegebene Wert * 10 Millisekunden.

```
CONST( pwmValue, duration )
```

Friert einen Helligkeitswert für „unendliche“ Zeit ein (Änderung nur durch Wechsel des Patterns):

```
HOLD( pwmValue )
```

Definiert eine lineare Änderung des PWM-Werts von einem Anfangswert zu einem Endwert innerhalb einer bestimmten Zeit:

```
RAMP( pwmValueStart, pwmValueEnd, duration )
```

Schaltet den PWM-Wert für eine bestimmte Dauer auf Null:

```
PAUSE( duration )
```

Darüber hinaus existieren die Makros `SEQUENCE` und `ARRAY` um die Lesbarkeit des Codes zu erleichtern.

Ein konstanter Ausgabewert mit dem Namen „constOn“ wird beispielweise so erzeugt:

```
SEQUENCE( constOn )  = { HOLD( 150 ) };
```

Ein Blitzer, der 800ms nach Sequenzstart für 200ms blitzt und dann 400ms Pause macht, wird mit dieser Zeile definiert:

```
SEQUENCE( singleFlash ) = { PAUSE(80), CONST(150,20), PAUSE(40) };
```

Grundsätzlicher Aufbau eines Minimalprogramms

Im ersten Schritt werden immer die Sequenzen wie im vorigen Abschnitt gezeigt, definiert.

Die Steuerung aller Funktionen erfolgt dann über das Objekt, das statisch außerhalb aller Funktionen definiert wird:

```
UniPWMControl ctrl;
```

In der Setup-Funktion wird das ctrl-Objekt initialisiert (hier 2 Kanäle) und die Sequenzen werden hinzugefügt und dem einzigen Pattern „0“ zugeordnet. Am Ende wird das Pattern aktiviert.

```
void setup()
{
  ctrl.Init( 2 ); // max 2 Kanäle
  ctrl.AddSequence( 0, PIN_TOP_STROBE, ARRAY( doubleFlash ) );
  ctrl.AddSequence( 0, PIN_BOT_STROBE, ARRAY( singleFlash ) );
  ctrl.ActivatePattern( 0 ); // wir haben nur ein Pattern, das aktivieren wir
```

```
}
```

Die loop-Funktion bleibt leer, bzw. wird per delay()-Befehl „gebremst“:

```
void loop()
{
  delay( 500 );
}
```

Komplexere Sequenzen

Das Beispiel zeigt eine etwas komplexere Sequenz mit zwei Rampen, die einem „Beacon-Licht“ ähneln. Die Sequenz sieht so aus:

```
SEQUENCE( beacon) = { RAMP(0,45,20), RAMP(45,200,20), CONST(200,20), RAMP(200,45,20),
RAMP(45,0,20), PAUSE(20) };
```

Die Aktivierung im Programm ist mit wenigen Zeilen erledigt:

```
void setup()
{
  ctrl.Init( 1 ); // ein Kanal
  ctrl.AddSequence( 0, PIN_TOP_STROBE, ARRAY( beacon) ); //Beacon Sequenz
  ctrl.ActivatePattern( 0 );
}
```

Umschalten von Patterns

Das Beispiel zeigt, wie man Sequenzen in Patterns organisiert und sie mittels der Fernsteuerung umschaltet. Das Beispiel zeigt das Ein- und Ausschalten eines Beacon-Lichts und einem Doppelblitzer. Neben dem Hinzufügen der Sequenzen und dem Zuordnen zu einem Pattern werden zwei Schalterpositionen definiert, die jeweils ein Pattern schalten. Die ersten beiden Werte beim Funktionsaufruf kennzeichnen den numerischen Wert des Eingangskanals. Die Werte müssen für den jeweiligen Empfänger anhand der Ausgabe im Serial-Monitor der Arduino-DIE ermittelt werden. Die DoLoop-Funktion ist notwendig, damit der empfangene Schalterwert zyklisch aktualisiert wird.

```
void setup()
{
  enum { PATTERN_FLIGHT, PATTERN_OFF };

  ctrl.Init( 10 );                      // max 10 output channels
  ctrl.SetInpChannelPin( RECEIVER_INP, UniPWMInpChannel::INP_INVERTED );

  // flight
  ctrl.AddSequence( PATTERN_FLIGHT, TOP_STROBE, ARRAY( beacon ) );
  ctrl.AddSequence( PATTERN_FLIGHT, BOT_STROBE, ARRAY( doubleFlash ) );

  // off
  ctrl.AddSequence( PATTERN_OFF, TOP_STROBE,    ARRAY( constOff ) );
  ctrl.AddSequence( PATTERN_OFF, BOT_STROBE,    ARRAY( constOff ) );

  // Schalter
  ctrl.AddInputSwitchPos( 1, 36, PATTERN_LANDING );
  ctrl.AddInputSwitchPos( 40, 46, PATTERN_FLIGHT );

  ctrl.ActivatePattern( PATTERN_FLIGHT );
}

void loop()
{
```

```
  ctrl.DoLoop();
  delay( 500 );
}
```

Batterieschutz

Der gezeigte Befehl schaltet beim Unterschreiten einer bestimmten Spannung auf ein bestimmtes Pattern um. Sind LEDs angeschlossen, die durch das Pattern `PATTERN` abgeschaltet werden, gehen diese bei zu niedriger Spannung aus. Die DoLoop()-Funktion muss hierzu wieder zyklisch in der loop()-Funktion aufgerufen werden.

```
ctrl.SetLowBatt( 755, PATTERN_OFF ); // 840 ~8.28 Volt, ~ 755 3.7V
```

Unterschreitet der Batteriestand den eingestellten Wert, wird neben der Umschaltung auf das angegeben Pattern, die CPU in den Sleep-Modus geschickt. Beim Arduino Nano bleibt der USB-Chip jedoch eingeschaltet, so dass die Batterie weiterhin mit mehreren Milliampere belastet wird. Um eine Tiefentaldung zu vermeiden sollte diese bald nach dem Umschalten abgeklemmt werden.

Servoanschaltung

Für die Anschaltung von Servos sind einige Einschränkungen zu beachten:

1. Die PWM-Frequenz beträgt im Standard 100Hz. Damit können nur digitale Servos angesteuert werden. Analoge Servos werden heiss und werden zerstört.
2. Der PWM-Bereich ist nicht auf die üblichen 1ms bis 2ms Pulsdauer beschränkt. Ungeeignete Pulswerte können ein Servo in den Anschlag fahren und den Motor blockieren lassen. Zulässige PWM-Werte müssen vorsichtig experimentell für das jeweilige Servo ermittelt werden.
3. Die Auflösung bei Software PWM beträgt nur ca. 40 Schritte. Der Jitter (Phasenrauschen) des Ausgabesignals kann bei manchen Servos zu leichtem Zittern führen.
4. An Pin 9 kann ein Analog-Servo angeschaltet werden, das in der Betriebsart ANALOG_SERVO verwendet werden muss. Die Ansteuerfrequenz beträgt dann 50 Hz, die Auflösung beträgt ca. 250 Schritte. Der Jitter ist in dieser Betriebsart sehr gering, so dass kein Servozittern zu befürchten ist.

Beispiele

Im Example-Verzeichnis der Library sind Beispiele zu finden:

- UniPWM_Sample1 – 2 LEDs Einfachblitz und Doppelblitz
- UniPWM_Sample2 – Beacon-Licht mit Onboard LED (benötigt keine externe Beschaltung)
- UniPWM_Sample3 – Vollständige LED-Steuerung mit Servo für Landescheinwerfer

Stromversorgung und Empfängeranschluss

In Flugmodellen und anderen sicherheitskritischen Anwendungen muss für die Schaltung unbedingt eine separate Stromversorgung verwendet werden. Alle Beispiele in diesem Dokument beziehen sich auf den Anschluss eines Lithium-Polymer-Akkus mit 2 Zellen (7,4Volt).

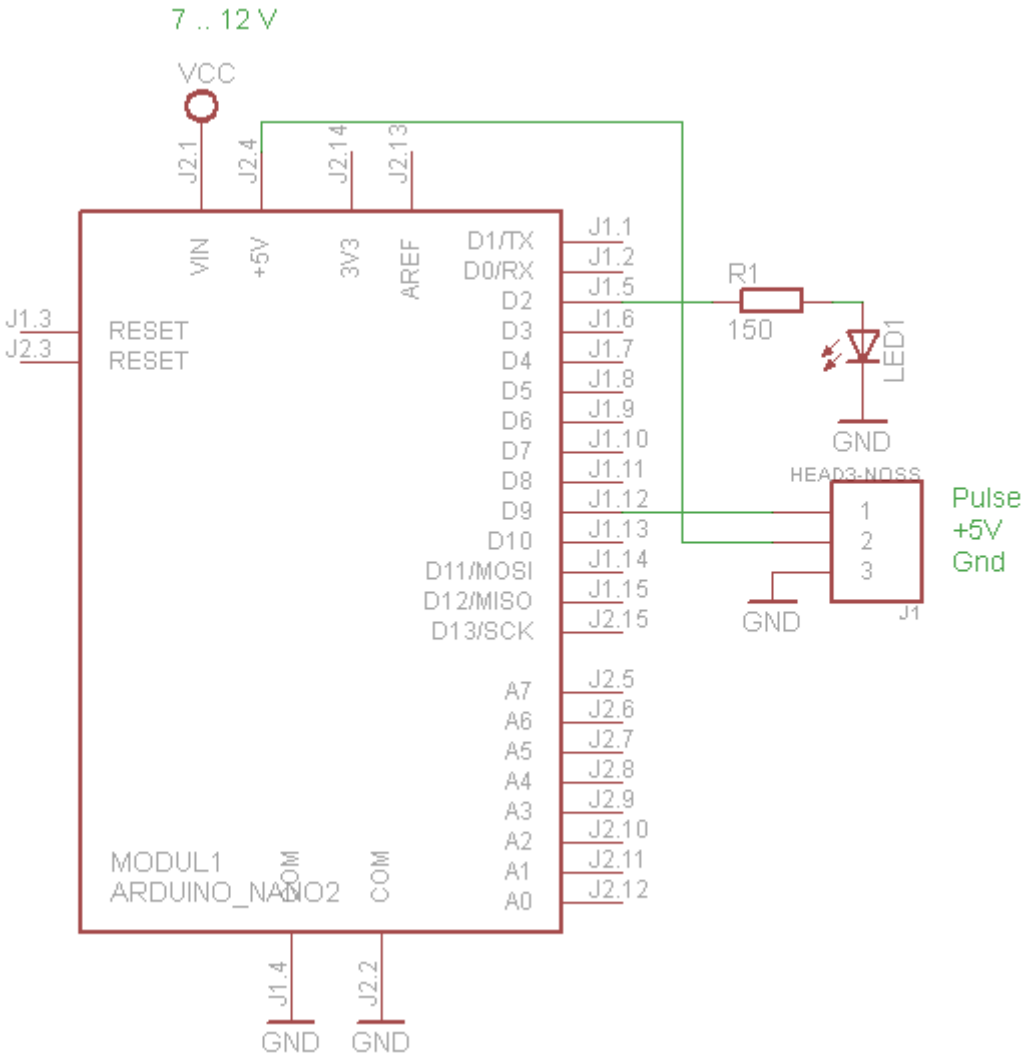
Ein RC-Empfänger muss in Flugmodellen stets über einen Optokoppler angeschlossen werden. Die Stromversorgung des Empfängers darf nicht mit der Schaltung verbunden werden.

Einfache Anschaltung von LEDs und Servos

Die Schaltung zeigt exemplarisch die Anschaltung einer LED und eines Servos direkt an den Arduino Nano ohne Treiberbaustein. Der Arduino Nano kann pro Ausgang 40 mA liefern. Es empfiehlt sich, superhelle LEDs mit max. 20 mA zu verwenden. Die mittlere Gesamtstromaufnahme aller LEDs sollte 200 mA nicht übersteigen.

Beim Anschluss des Servos ist zu beachten, dass dieses keine hohen Lasten bewegen oder blockieren darf, da erhöhter Stromverbrauch den Prozessor zum Absturz bringt.

Wird mehr Strom benötigt oder soll ein RC-Empfänger angeschlossen werden, sollte die erweiterte Anschaltung (Treiber und Optokoppler) verwendet werden.



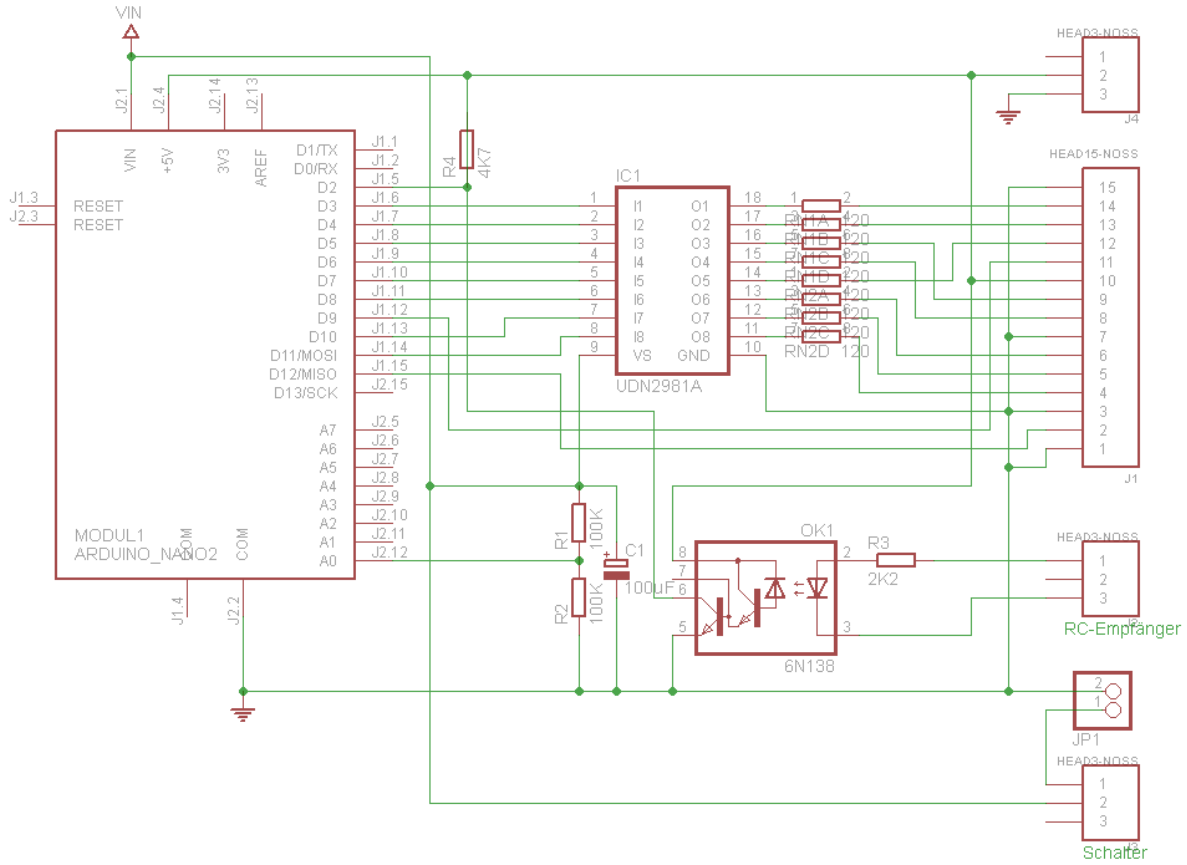
Erweiterte Anschaltung von LEDs und Servos

Die erweiterte Anschaltung ist optimiert auf den Anschluss von 8 LEDs an einem Darlington-Treiber, der bis zu 500 mA pro Ausgang liefern kann. Damit wird die Anschaltung von LED-Emittern (z.B. für Landescheinwerfer möglich). Ein RC-Empfänger wird über einen Optokoppler angeschlossen.

Die Stromversorgung wird über einen Analogeingang überwacht, bei niedriger Versorgungsspannung können die Verbraucher softwaregesteuert abgeschaltet werden. Theoretisch kann durch diese Messung auch die Helligkeit der LEDs an Schwankungen in der Versorgungsspannung angepasst werden (→digitale Stromquelle, derzeit in der Software nicht implementiert).

An Pin 9 ist ein analoges Servo anschliessbar (50Hz).

Die Vorwiderstände sind mit ca. 120 Ohm dimensioniert. Bei einer Versorgungsspannung von 7,4 Volt, einer Sättigungsspannung der Treiberstufen von 1,6 Volt und einer Vorwärtsspannung der LED von ca. 2,2 Volt ergibt sich ein maximaler Strom von 30 mA. Der mittlere Strom durch die LED kann durch die Helligkeitssteuerung eingestellt werden, so dass der Wert (auf Kosten der Auflösung der Helligkeitssteuerung) unkritisch ist.



Anhang

Platine der erweiterten Anschaltung.

Folgt …