

Infrarot Lichtschranke mit Arduino zum Auslesen des Stromzählers

Martin Kompf



Elektromagnetische Ferraris-Stromzähler kommen noch in vielen Haushalten zur Verbrauchsmessung zum Einsatz. Sie besitzen keine direkte Schnittstelle zur elektronischen Datenerfassung. Auf der anderen Seite hilft jedoch eine präzise und minutengenaue Erfassung des Stromverbrauchs beim Energiesparen, da sich so Standbyverbrauch und Lastspitzen besser analysieren lassen. Eine Möglichkeit, dennoch an diese Daten zu kommen, besteht in der optoelektronischen Abtastung der Zählscheibe. Die Signalerfassung erledigt dabei ein Arduino Nano.


ALMASOLAR



**Solarmodul
l'M Premium
270W**

159€
MWST INKL.



Ferraris-Zähler

Der **Ferraris-Zähler** arbeitet nach dem Prinzip eines Motors. Der Stromfluss durch mehrere Spulen versetzt eine Zählscheibe in Drehung, die ihrerseits ein mechanisches Zählwerk antreibt. Die Anzahl der Umdrehungen der Scheibe ist dabei proportional zur verrichteten elektrischen Arbeit, die typischerweise in Kilowattstunden (kWh) gemessen und abgerechnet wird. So steht zum Beispiel auf meinem Zähler die Angabe 75 U/kWh, das heißt 75 Umdrehungen der Zählscheibe bedeuten einen Verbrauch von 1 kWh.

Auf der Zählscheibe ist eine rote Markierung angebracht. Mittels einer Reflex-Lichtschranke lässt sich der Durchlauf dieser Markierung durch das Sichtfenster erfassen und elektronisch weiterverarbeiten.

Arduino Lichtschranke

Die eigentliche Lichtschranke besteht aus einer Infrarot-Leuchtdiode **SFH 409** (alternativ: **SFH 4346**) und einem Infrarot-Fototransistor **SFH 309 FA** (Abb. 1). Die Ansteuerung erfolgt mit einem **Arduino Nano**. Prinzipiell sind natürlich auch andere Modelle des **Arduino** geeignet, der Nano kommt hauptsächlich wegen seiner geringen Abmessungen zum Einsatz. Die Anschlussbezeichnungen in Abb. 1 beziehen sich direkt auf die Beschriftung des Arduino Boards.

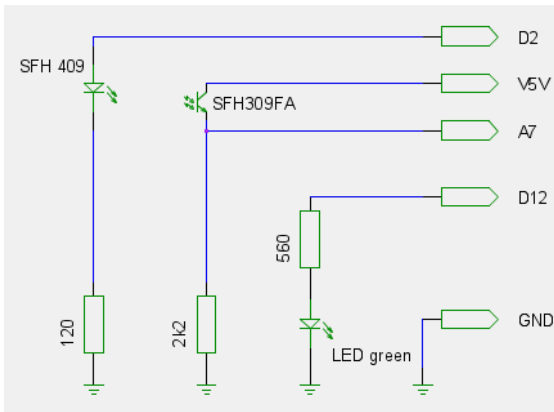


Abb. 1: Schaltbild der Infrarot-Reflex-Lichtschranke

Gegenüber einem «herkömmlichen» Aufbau einer Lichtschranke mit analogen Triggerschaltkreisen oder gar diskreten Transistoren bietet die Verwendung eines Mikrocontrollers viele Vorteile. So sind neben den beiden Infrarot-Bauelementen nur zwei weitere passive Bauteile notwendig: Ein Widerstand von 120 Ω dient zur Begrenzung des Stroms durch die Leuchtdiode und der Widerstand 2,2 k Ω wandelt den durch den Fototransistor fließenden Strom in eine Spannung um. Sie gelangt direkt an den Analogeingang **A7** des Arduino.

Die Anode der Leuchtdiode ist nicht direkt mit der Betriebsspannung **V5V**, sondern mit dem Digitalausgang **D2** verbunden. Dadurch lässt sich eine wirksame Unterdrückung von störender Umgebungsstrahlung erreichen. Das Messprogramm schaltet sehr schnell über **D2** die Leuchtdiode an und aus und misst jeweils die an **A7** entstehende Spannung. Die anschließende Differenzbildung eliminiert das auf den Sensor treffende

Umgebungslicht, das Resultat enthält ausschließlich das von der Leuchtdiode erzeugte Licht. Dieses Messprinzip ließe sich ohne einen Mikrocontroller nur sehr aufwändig realisieren.

Die beiden weiteren Bauelemente LED grün und Widerstand 560 Ω sind für die eigentliche Funktion der Lichtschranke nicht erforderlich. Sie dienen ausschließlich zur visuellen Rückmeldung der korrekten Funktion des Gerätes. Abgleichwiderstände oder Potentiometer sucht man in der Schaltung vergeblich. Die Einstellung der Triggerpegel zur Anpassung der Lichtschranke an unterschiedliche Umgebungen erfolgt per Software.

Der Preis für einen Arduino Nano bewegt sich zwischen 8 US \$ für einen «China-Klon» und 50 €. Beim Einkauf lohnt daher ein Preisvergleich! Man muss auch keinen Nano nehmen, wenn man genügend Platz hat, lässt sich durchaus ein anderes Modell verwenden.

Aufbau

Die Bauteile inklusive des Arduino Nano sind auf einer Universal-Lochrasterplatte verlötet. Ihre Größe ist so gewählt, dass sie ohne weitere Befestigungselemente «saugend» in ein Standard-Plastikgehäuse passt (Abb. 2). Der Arduino ist dabei so positioniert, dass sein Mini-USB-Anschluss auch bei geschlossenem Gehäuse später noch von außen zugänglich ist.

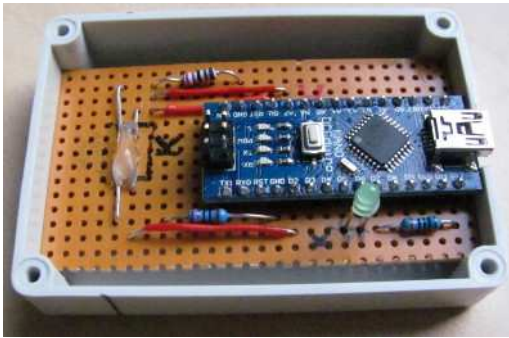


Abb. 2: Der Arduino Nano und die wenigen externen Bauelemente kommen in kleines Plastikgehäuse.

Infrarot-Leuchtdiode und -Fototransistor sind von der Unterseite der Platine bestückt (Abb. 2 links), damit das Infrarotlicht durch zwei in die untere Gehäuseschale gebohrte Löcher nach außen gelangen kann.



Abb. 3: Das Gehäuse von außen. Man erkennt Infrarot-Leuchtdiode und -Fototransistor. Die Markierung hilft später bei der exakten Ausrichtung der Lichtschranke auf dem Zähler.

Die Oberseite der Bauelemente liegt dann später genau auf Oberfläche des Zählergehäuses direkt über der Zählscheibe (Abb. 4). Die mit einem Permanentmarker aufgebrachte, auf die Gehäuseseiten verlängerte Markierung erleichtert die Positionierung auf dem Zähler (Abb. 3). Damit die Reflex-Lichtschranke funktioniert, müssen Leuchtdiode und Fototransistor so eng wie möglich beieinander sitzen, ohne dass jedoch direktes Licht von der Diode auf den Transistor treffen kann.

Die grüne Leuchtdiode ist normal von oben bestückt, sodass sie später auf der vom Zähler abgewandten Seite sichtbar ist (Abb. 2 unten rechts). Ihre Länge ist so gewählt, dass sie mit der Gehäuseoberseite abschließt.

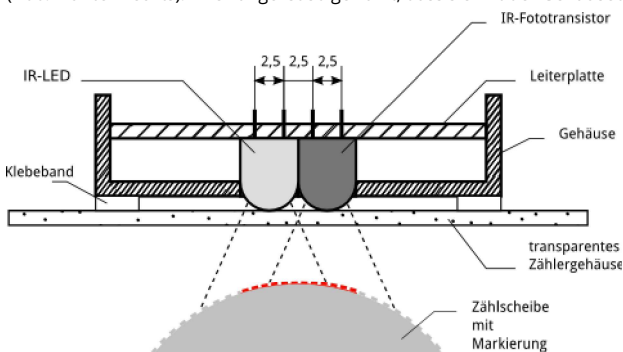


Abb. 4: Schnittansicht der Reflex-Lichtschranke (schematisch)

Software

Ein Arduino eignet sich zwar hervorragend zur Ansteuerung der Lichtschranke und Echtzeitdatenerfassung, hat jedoch seine Grenzen, wenn es an die Langzeitspeicherung und Visualisierung der Daten geht. Aus diesem Grund besteht die Software aus zwei Teilen. Ein Teil läuft auf dem Arduino und kümmert sich um die Datenerfassung. Den zweiten Teil kann man auf einem beliebigen (vorzugsweise Linux-) Rechner installieren, der über einen permanenten Massenspeicher (zum Beispiel eine Festplatte oder SD-Speicherkarte) sowie über USB- und Netzwerkschnittstelle verfügt. Eine gute Wahl ist ein [Raspberry Pi](#). Da sich in meinem Keller schon ein RasPi zur [Messung des Gasverbrauchs](#) befindet, habe ich gleich diesen verwendet. Arduino und Steuerrechner sind über ein USB-Kabel verbunden. Dieses gewährleistet zum einen die Stromversorgung des Arduino und zum anderen erfolgt hierüber die Kommunikation zwischen beiden Softwareteilen mittels eines einfachen seriellen Protokolls.

Die komplette Software steht auf [Github](#) im Repository **emeir** (electric meter with infrared light barrier) zur Verfügung. Eine Kopie gelangt per

```
git clone https://github.com/skaringa/emeir.git
```

in das lokale Arbeitsverzeichnis.

Die Datei **arduino_sketch/ReflectorLightBarrier.ino** enthält den Arduino-Sketch. Er ist mittels der [Arduino-IDE](#) zu kompilieren und in den Programmspeicher des Arduino zu laden.

Nach dem Start ist die Arduino Software im *Trigger data mode*. Für einen ersten Funktionstest besser geeignet ist der *Raw data mode*, in dem die Software ununterbrochen die am Fototransistor gemessene Differenzspannung auf der seriellen Schnittstelle ausgibt. Die Umschaltung des verschiedenen Modi erfolgt durch das Senden von Kommandos vom Steuerrechner zum Arduino. Dazu kann man den Serial Monitor der Arduino-IDE oder ein Terminalprogramm wie *minicom* verwenden.

Das Senden des Zeichens **C** zum Arduino veranlasst diesen, in den Kommandomodus zu gehen. Nun reagiert er auf verschiedene Steuerbefehle:

```
D - Raw data mode: Die Software sendet fortlaufend Messwerte.
T - Trigger data mode: Nur bei Umschaltung des Triggers wird eine 1 oder 0 gesendet.
S low high - Setzen der Triggerpegel. Parameter sind zwei Integer-Werte von 0 bis 1023.
C - Aus den beiden Datenmodi gelangt man stets mittels C wieder in den Kommandomodus.
```

Im *Raw data mode* gibt der Arduino ständig Messwerte auf der seriellen Schnittstelle aus. Dabei handelt es sich um den Differenzwert, den die Software wie oben beschrieben durch Subtraktion der Spannungen im ein- und ausgeschalteten Zustand der IR-Leuchtdiode berechnet. So kann man testen, ob beim Aufbau der Hardware alles glatt gegangen ist: Hält man ein Stück Papier vor die Reflex-Lichtschranke, dann sollte die Veränderung seines Abstands signifikante Unterschiede in den Messwerten bewirken.

Der für die Messwerterfassung relevante Programmausschnitt ist:

```
// perform measurement
// turn IR LED off
digitalWrite(irOutPin, LOW);
// wait 10 milliseconds
delay(10);
// read the analog in value:
sensorValueOff = analogRead(analogInPin);
// turn IR LED on
digitalWrite(irOutPin, HIGH);
delay(10);
// read the analog in value:
sensorValueOn = analogRead(analogInPin);

switch (dataOutput) {
  case 'R':
    // print the raw data to the serial monitor
    Serial.println(sensorValueOn - sensorValueOff);
    break;
  case 'T':
    // detect and output trigger
    detectTrigger(sensorValueOn - sensorValueOff);
    break;
}
```

Montage und Abgleich

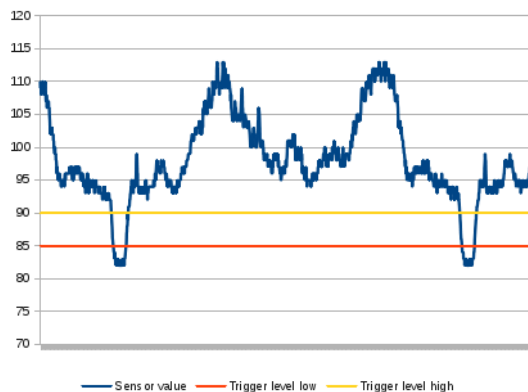


Abb. 5: Die gemessenen Sensorwerte (blau) beim zweimaligen Durchlauf der Zählmarkierung und die daraus bestimmten Triggerschwellen.

90 (high, gelb).

Das Kommando `S 85 90` schreibt die Werte für die Triggerschwellen in den EEPROM des Arduino. Sie überleben damit das Abtrennen der Stromversorgung und einen Programmneustart.

Von nun an betreibt man die Arduino-Software im *Trigger data mode*. Hier erfolgt eine Ausgabe auf der seriellen Schnittstelle nur noch bei einem Triggerereignis: Bei Unterschreitung des *low* Triggerlevels gibt das Programm das Zeichen 0 und bei Überschreitung des *high* Levels eine 1 aus. Außerdem wird die grüne Leuchtdiode entsprechend geschaltet, was für eine visuelle Kontrolle der Funktion enorm hilfreich ist: Nur wenn sich die rote Zählmarkierung vor der Lichtschranke befindet, ist der Triggerzustand «0» und die Leuchtdiode ist aus, ansonsten leuchtet sie:

```
/**
 * Detect and print a trigger event
 */
void detectTrigger(int val) {
  boolean nextState = triggerState;
  if (val > triggerLevelHigh) {
    nextState = true;
  } else if (val < triggerLevelLow) {
    nextState = false;
  }
  if (nextState != triggerState) {
    triggerState = nextState;
    Serial.println(triggerState? 1 : 0);
    // control internal LED
    digitalWrite(ledOutPin, triggerState);
  }
}
```

Die Lichtschranke ist mit zwei Streifen doppelseitigem Klebeband auf dem Zählergehäuse befestigt (siehe Bild am Artikelanfang). IR-Leuchtdiode und Fototransistor müssen dabei genau über der Zählscheibe sitzen. Bei der genauen Positionierung hilft die auf dem Gehäuse angebrachte Markierung (Abb. 3).

Bei laufendem Stromzähler erfolgt zunächst die Erfassung der Messdaten im *Raw data mode*. Hilfreich ist hier ein Terminalprogramm wie zum Beispiel *minicom*, das die über die serielle Schnittstelle laufenden Daten in eine Textdatei schreiben kann. Nach dem Import dieser Datei in ein Tabellenkalkulationsprogramm (zum Beispiel *LibreOffice Calc*) kann man eine Grafik ähnlich wie Abb. 5 erzeugen, die bei der Ermittlung der Triggerschwellen hilft: Im zeitlichen Verlauf der Messdaten (blau) sind eindeutig die beiden Durchläufe der Zählmarkierung zu identifizieren, denn deren rote Farbe führt zu einer signifikanten Abnahme des reflektierten Lichts. Hier ergibt sich daraus die Festlegung der beiden Triggerschwellen auf 85 (low, rot in Abb. 5) und

Aufzeichnung von Zählerstand und Verbrauch

Auf dem Steuerrechner läuft das Python-Skript **emeir.py**, welches über die USB-Seriell Schnittstelle die vom Arduino gelieferten Daten empfängt. Im *Trigger data mode* sind das nur Nullen und Einsen, deren Umschaltung den Durchlauf der Zählmarkierung signalisiert. Da das Verhältnis von Umdrehungen der Zählscheibe zum Stromverbrauch bekannt ist, kann man somit direkt auf die verbrauchte Energiemenge schließen:

```
# Serial port of arduino
port = '/dev/ttyUSB0'

# Revolutions per kWh
rev_per_kWh = 75

# ...

# Open serial line
ser = serial.Serial(port, 9600)

trigger_state = 0
counter = last_rrd_count()
trigger_step = 1.0 / rev_per_kWh

while(1==1):
    # Read line from arduino and convert to trigger value
    line = ser.readline()
    line = line.strip()

    old_state = trigger_state
    if line == '1':
        trigger_state = 1
    elif line == '0':
        trigger_state = 0
    if old_state == 1 and trigger_state == 0:
        # trigger active -> update count rrd
        counter += trigger_step
        update = "N:%.2f:%.0f" % (counter, trigger_step*3600000.0)
        rrdtool.update(count_rrd, update)
```

Das Programm schreibt bei einer Auslösung des Triggers den neuen Zählerstand und die seit dem letzten Trigger verbrauchte Energie $\text{trigger_step} = 1 / 75$ (Umdrehungen pro kWh) in eine Round Robin Datenbank. Dieses Prinzip gleicht dem in [Gaszähler auslesen mit Magnetometer HMC5883 und Raspberry Pi](#) beschriebenen Vorgehen.


Das Programm übernimmt auch das Anlegen der Datenbank **emeir.rrd**, die im gleichen Verzeichnis wie das Python-Skript liegt. Hierzu muss man es einmalig mit der Option -c starten:

```
./emeir.py -c
```

Eine neu angelegte Datenbank startet logischerweise mit dem Zählerstand 0. Um ihn mit dem mechanischen Zähler zu synchronisieren, notiert man sich den mechanischen Zählerstand (zum Beispiel 132290.0) und schreibt diesen möglichst zeitnah in die Datenbank:

```
rrdtool update emeir.rrd N:132290.0:0
```

Danach ist **emeir.py** neu zu starten. Es liest dabei den gespeicherten Zählerwert aus der Datenbank und setzt damit die Zählung fort.

Mittels **rrdtool**  lassen sich aus der Round Robin Datenbank ansprechende Grafiken erzeugen. Die folgenden Kommandos erzeugen die in Abb. 6 gezeigten Grafiken über den Zählerstand und Verbrauch der letzten drei Tage:

```
# Erzeuge Zählerstandsgrafik in counter.gif
rrdtool graph counter.gif \
-s 'now -3 day' -e 'now' \
-X 0 -Y -A \
DEF:counter=emeir.rrd:counter:LAST \
LINE2:counter#000000:"Zählerstand [kWh]"
# Erzeuge Verbrauchsgrafik in consum.gif
rrdtool graph consum.gif \
-s 'now -3 day' -e 'now' \
-Y -A \
DEF:consum=emeir.rrd:consum:AVERAGE \
LINE2:consum#00FF00:"Verbrauch [W]"
```

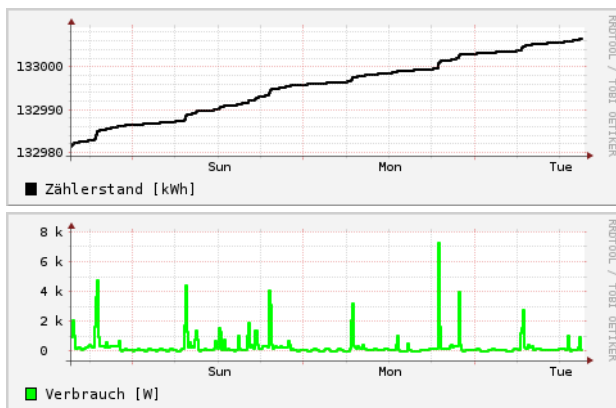


Abb. 6: Mit rrdtool erzeugte Grafiken von Zählerstand und Stromverbrauch über drei Tage

Ein schlanker Webserver wie [lighttpd](#) «lighty» zusammen mit ein paar HTML-Seiten und Perl-Skripts stellt die Verbrauchsgrafiken komfortabel im Intranet zur Verfügung. Das Prinzip ist in den Beiträgen zur [Erfassung des Gasverbrauchs](#) und [Temperaturmessung](#) beschrieben.

Fazit

Das kleine, selbstgebaute Gerät läuft seit über einem Jahr störungsfrei und lieferte schon interessante Erkenntnisse. Signifikant ist zum Beispiel der Unterschied zwischen Urlaubs- und Anwesenheitszeiten. Deswegen sollte man auch sehr vorsichtig bei der Entscheidung sein, wem man seine Verbrauchsdaten in Echtzeit zur Verfügung stellt! Neben den Verbrauchsspitzen, für die hauptsächlich der elektrisch betriebene Küchenherd und die Warmwasserbereitung verantwortlich sind, fällt die leicht wellige Grundlast auf. Das ist dem Betrieb von Kühlschrank beziehungsweise Kühltruhe geschuldet, die sich periodisch ein- und ausschalten. Zieht man dies ab, dann bleibt ein Grundverbrauch von etwa 60 W übrig, der durch den Betrieb diverser elektronischer Geräte, wie DSL-Router, Radiowecker und natürlich auch den für die Verbrauchsmessung benutzten Arduino und Raspberry Pi (!) entsteht. Eine weitere Eingrenzung erfordert dann den Einsatz eines portablen Energiekosten-Messgeräts, das man in die Netzleitungen der einzelnen Geräte einschleift. So etwas gibt es für ambitionierte Selbismacher auch als [Bausatz von ELV](#).

Interessante Links

[Programmcode](#) - Repository auf Github

[OpenCV Praxis: OCR für den Stromzähler](#) - Alternativprojekt, das den Stromzähler mit USB-Kamera und optischer Zeichenerkennung ausliest!

[Arduino](#) - Offizielle Seite zu Hard- und Software

[Raspberry Pi](#) Wiki bei eLinux.org

[Raspberry Pi Foundation](#)

[RRDtool](#) - OpenSource Datenerfassungs- und -visualisierungstool


ALMASOLAR



**Solarmodul
l'M Premium
270W**

159€
 MWST INKL.

[Bestellen >](#)