Vgeiger – Ein virtueller Geigerzähler auf Arduino Basis zum Testen von Elektronik zur Zählraten-Messung mit vorgegebener Poisson-Statistik

Bernd Laquai, 5.12. 2023

Wenn man einen Geigerzähler selbst baut, dann wird man heutzutage das Berechnen der Impulsrate in der Regel einem Mikrocontroller überlassen. In aller Regel liefert der Impulsverstärker des Zählrohrmoduls bereits einen digitale Zählimpulse. Da jedoch die Zählimpulse völlig unvorhersehbar eintreffen, führt man das digitale Ausgangssignal des Detektor-Moduls üblicherweise auf den Interrupt-Eingang des Mikrocontrollers, damit das Auftreten eines Zählimpulses als asynchroner Interrupt gewertet werden kann. Die normale Programmausführung wird beim Auftreten eines Interrupts unterbrochen und es wird in eine Interruptroutine verzweigt, welches die Impulse zählt. Nach der Registrierung eines Zählpulses durch Hochzählen eines Zählers kehrt die Programmausführung dann wieder aus der Interruptroutine zur normalen Programmausführung zurück, in der beispielsweise die Anzeige aufgefrischt wird, sobald das Zählintervall abgelaufen ist.

Allerdings muss die Interruptroutine möglichst kurzgehalten werden, da sie ja eine Totzeit erzeugt, in der normalerweise kein weiterer Zählimpuls registriert werden kann. Das aber wirft die Frage auf: Ist denn der Mikroconroller auch schnell genug für eine gewisse Aktivität, die man messen möchte?

Nun wissen wir aus statistischen Betrachtungen /1/, dass der zeitliche Abstand der Zählimpulse beim Geigerzähler negativ exponentiell verteilt ist, d.h. dass kurze Abstände viel häufiger auftreten, und ein Abstand umso wahrscheinlicher ist, je kürzer er ist. Um einen Mikrocontroller und die Software dafür zu testen, ob sie Zählimpulse, die in kurzem Abstand aufeinanderfolgen auch getrennt wahrnehmen kann, kann man natürlich einen handelsüblichen Pulsgenerator an den Interrupteingang anschließen und periodische Impulse mit variabler Frequenz erzeugen und so ausloten, wann die ermittelte Zählrate gegenüber der eingestellten Frequenz zu sehr abnimmt. Aber das ist möglicherweise zu optimistisch, da es davon ausgeht, dass der Zählimpulsverlust bei einer gewissen, hohen Rate gerade noch erträglich ist, aber wenn die Zählimpulse wirklich zufällig wären, dann wäre der Mittelwert zwar niedriger, aber ein Großteil der Impulse würde in kürzerem Abstand eintreffen und könnten damit verschluckt werden.

Ein realistischer Test der maximalen Zählrate bei welcher der Impulsverlust noch erträglich ist, ist nur möglich, wenn man beim Zählen die Test-Zählimpulse auch mit der richtigen Statistik, also einer negativ exponentiellen Abstandsverteilung und bekannter mittlerer Impulsrate erzeugt. Wenn man die Statistik verstanden hat, dann kann man sich so einen virtuellen Geigerzähler, welcher solche Impulse synthetisch und ohne Detektor mathematisch korrekt erzeugt, mit ganz einfachen Mitteln in einen Mikrocontroller wie den Arduino programmieren.

Was man dazu braucht ist lediglich eine Routine, die gleichverteilte Zufallszahlen erzeugt und eine Mathe-Bibliothek, welche den Logarithmus berechnet. Wie in /2/ erklärt, berechnet man zunächst gleichverteilte Zufallszahlen zwischen 0 und 1 mit Hilfe einer Zufallsroutine. Dann benutzt man die inverse Verteilungsfunktion der Zufallsverteilung die man haben will und geht mit den zwischen 0 und 1 gleichverteilten Werten in diese inverse Verteilungsfunktion und erhält so Zufallszahlen nach der gewünschten Verteilungsfunktion. In unserem Fall ist die gewünschte Verteilungsfunktion der Abstände zwischen den Zählimpulsen durch die negativ exponentielle Verteilung vorgegeben:

$F(x) = 1-\exp(-\lambda x)$ für $x >= 0$, und 0 sonst

Dabei ist der Mittelwert $\mu = 1/\lambda$ (die mittlere Zeit zwischen den Zählimpulsen), wenn λ die mittlere Zählrate ist.

Die invertierte Verteilungsfunktion dazu ist dann:

$x = -\log_n(1-F)/\lambda$

Wenn also F gleichverteilt zwischen 0 und 1 ist, dann ist x negativ exponentiell verteilt mit der mittleren Zählrate von λ. Nun kann man mit dem Mikrocontroller die Abstände x einer Routine übergeben, die einfach nur diese Zeit wartet und danach ein Ereignis ausgeben, also beispielsweise einen Impuls. Wenn man das in einer Schleife macht, erzeugt man so einen Poisson-Prozess mit dem Parameter λ. Gibt man diese Impulse dann auf den zählenden Mikrocontroller, den man testen will, dann muss dieser eine Zählrate von λ anzeigen, wenn er richtig funktioniert.

Wählt man eine hohe Zählrate zum Beispiel λ = 100cps dann merkt man, sobald man in der Interruptroutine versucht mehr zu machen als nur zählen, dass dann der zählende Mikrokontroller Impulse verliert und daher eine geringere Zählrate als λ anzeigt.

Was für den Arduino Uno noch etwas speziell ist, ist die Art und Weise wie man von den Zufallszahlen der Routine random() zu den Zufallszahlen, die zwischen 0 und 1 gleichverteilt sind kommt. Man kann der Routine eine Zahl mitgeben, welche die obere Grenze der Zufallszahlen angibt, welche im long Format zurückkommen. Diese Zahl sollte so groß wie möglich sein, damit man das Intervall von 0 bis 1 mit möglichst vielen Werten füllen kann. Aus irgendeinem Grund ist aber die größte Zahl, die man beim Arduino Uno verwenden kann 2^24, dann sind die Zahlen, welche random() zwischen 0 und 2^24-1 verteilt. Das ist ausreichend groß. Dividiert man nun durch 2^24-1, dann sind die Zufallszahlen als double Zahlen zwischen 0 und 1 schön fein verteilt und das ist was man braucht.

Nachdem man mit delay() den entsprechenden Zeitabstand in Millisekunden gewartet hat, kann man einen digitalen Pin zum Beispiel HIGH und nach 5ms wieder auf LOW schalten um zum Beispiele einen Piezo Beeper mit Elektronik kurz klicken lassen. Allerdings erzeugt das einen Offset für den Abstand der Zählimpulse, allerdings sind 5ms wenig, wenn die mittlere Zählrate z.B. 10cps beträgt. Bei einer Rate von 100cps dagegen merkt man den 5ms Offset aber schon an der Statistik. Für einen echten Test einer Auswerteelektronik, sollte man daher den delay() Aufruf auskommentieren.

Literatur

/1/ Bernd Laquai; Die Statistik des Zerfalls
http://opengeiger.de/StatistikDesZerfalls.pdf

/2/ Bernd Laquai; Der virtuelle Geigerzähler und die Monte-Carlo Methode als wissenschaftliches Überbleibsel des Manhattan-Projekts
http://opengeiger.de/MonteCarloPoisson.pdf

Anhang

Virtueller Geigerzähler als Impulsgenerator für Impulse mit negativ exponentieller Abstandsverteilung und Poisson-verteilter Statistik in einer vorgegebenen Messzeit.

```
/*generiert ein Poisson verteiltes Geigerzähler Signal an Pin 5 mit
einer Impulsrate von lambda pro Millisekunde und eine Totzeit von 5
Millisekunden in der ein Piezo Beeper mit Elektronik aktiviert wird.
*/

void setup() {
  Serial.begin(9600);
  pinMode(5,OUTPUT);
  digitalWrite(5,LOW);
}

void loop() {
  //double lambda = 0.1; //100cps, auskommentieren von delay(5)!
  double lambda = 0.005; //5cps
  unsigned long rnl = random(16777216);
  double rnd = rnl/16777215.00;
  int negEx = round(-log(1-rnd)/lambda);
  //Serial.println(negEx);
  delay(negEx);
  digitalWrite(5,HIGH);
  delay(5);
  digitalWrite(5,LOW);
}
```

Messprogramm für die Zählrate eines Geigerzählers mit Impulsvorwahl. Diese Routine läuft auf einem Arduino Uno und soll beispielshaft mit dem obigen Impulsgenerator getestet werden. Gemessen wird die Zeit bis eine gewisse Anzahl (MAXINT) an Impulsen eingetroffen ist, aus Messzeit und Impulsanzahl wird die Zählrate ermittelt.

```
#define MAXCNT 100
volatile int counter = 0;
unsigned long oldTime = 0;

void count()
{
  counter++;
}

void setup(){
Serial.begin(9600);
attachInterrupt(digitalPinToInterrupt(2), count, FALLING);
}

void loop(){
  unsigned long time;
  unsigned long dt;
  float rate;
  int err;
  if (counter == MAXCNT) {
    detachInterrupt(digitalPinToInterrupt(2));
    time = millis();
    dt = time-oldTime;
```

```
    rate = (float)MAXCNT*1000.0/(float)dt;
    Serial.println(round(rate));
    oldTime = millis();
    counter = 0;
    attachInterrupt(digitalPinToInterrupt(2), count, FALLING);
  }
}
```