Rauschgenerator mit AVR

Für Untersuchungen im Audiobereich benötigte ich einen Rauschgenerator.

Ein Versuch mit der klassischen Lösung: Rauschende Z-Diode / Transistor viel nicht überzeugend aus: der Anteil an niedrigen Frequenzen war unbefriedigend.

Nach einigen Experimenten wurde die Schieberegister-Methode (Linear Feedback Register LFR) benutzt. Diese war in einer alten Version der Halbleiter-Schaltungstechnik von Tietze-Schenk beschrieben, ist aber leider in der neuen Auflage nicht mehr enthalten.

Eine interessante Information dazu findet sich hier:

http://www.xilinx.com/support/documentation/application_notes/xapp052.pdf

Bei dieser Methode wird ein Schieberegister mit mehreren Rückkopplungen benutzt.

Firmware

Es wird ein 32bit-Schieberegister mit Rückkopplung benutzt.
Von den 32 Bit werden letztendlich aber nur 8 Bit für die Ausgabe des Signals benutzt.

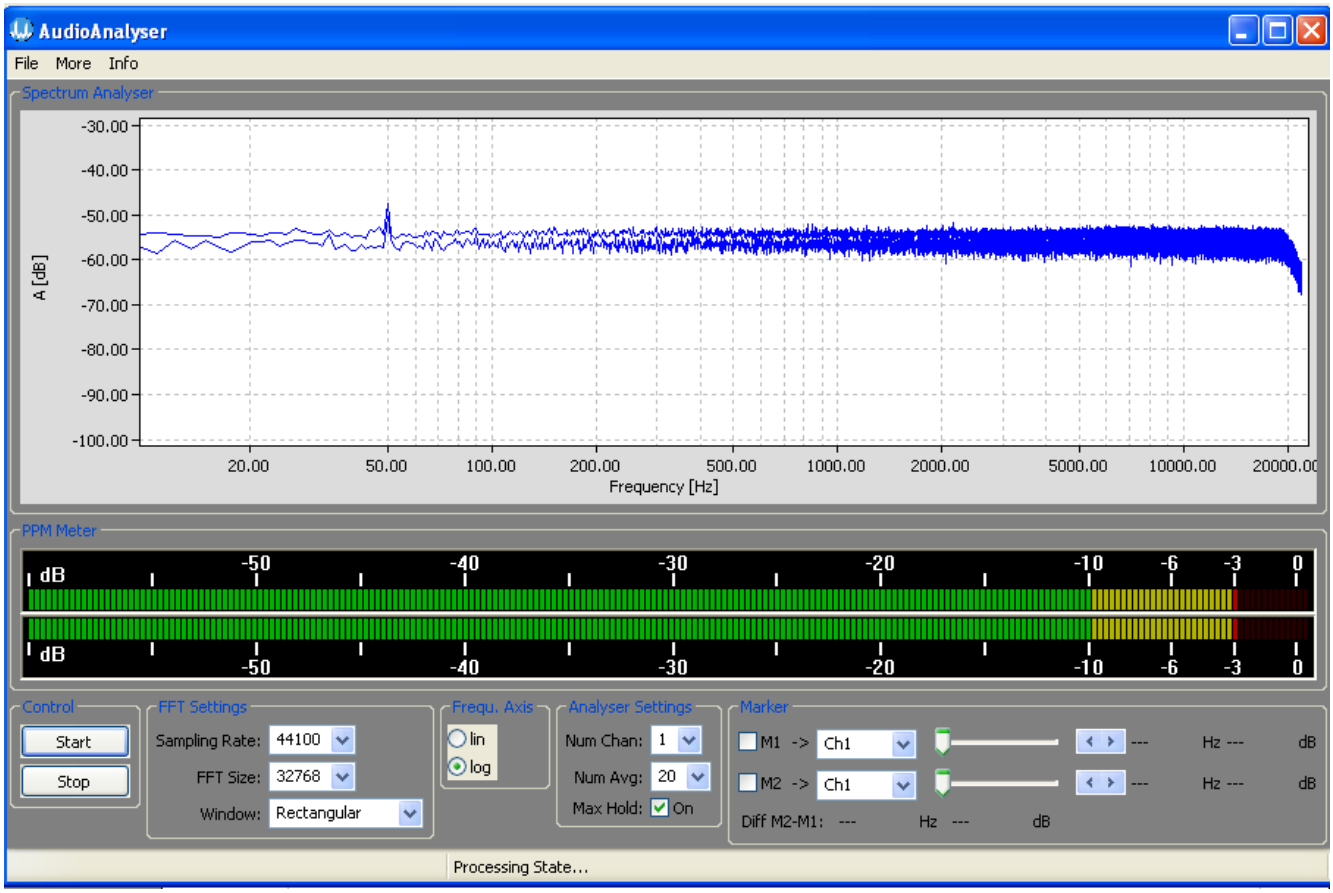
```
'Rauschgenerator mit der Schieberegister-Methode
'32 bit

    $regfile = "m8def.dat".dat "
    Config Portd = Output

    $asm

    LDI R21,0
    LDI R22,0           ;fuer XOR benutzt

    ldi R16, &h5C        ;Anfangswerte
    LDI R17, &hD3
    LDI R18, &hBC
    LDI R19, &HB4

0:
    LDI R20,32          ;4*8 = 32 bit
1:
    ROL R16             ;rotieren 3 Bytes ueber Carry
    ROL R17
    ROL R18
    ROL R19

    'Vorsicht: Bitnummerierung in der Literatur geht von 1..n
    'nicht von 0...n-1
    BST R19,7           ;Bit 32 = Bit 7 in R19 -> T-Flag
    BLD R21,0             ;dieses vom T-Flag in R21, Bit 0
    BST R18,5           ;Bit 22 = Bit 5 in R18 -> T-Flag
    BLD R22,0             ;dieses vom T-Flag in R22, Bit 0
    EOR R21,R22            ;XOR
    BST R16,2           ;Bit 2 = Bit 1  in R16 -> T-Flag
    BLD R22,0             ;dieses vom T-Flag in R22, Bit 0
    EOR R21,R22            ;XOR
    BST R16,0           ;Bit 1 = Bit 0  in R16 -> T-Flag
    BLD R22,0             ;dieses vom T-Flag in R22, Bit 0
    EOR R21,R22            ;XOR
    BST R21,0             ;R21, Bit 0 -> T-Flag
    BLD R16,0            ;T-Flag -> R16, Bit 0

    !Out Portd , R18

    DEC r20             ;schon fertig?
    BRNE 1              ;wenn nicht, weiter

    RJMP 0
```

Hardware

Im Gegensatz zum Listing wurde aus Recycling-Gründen ein 2313-AVR benutzt, der hier vollauf genügt. Die Ausgabe des analogen Signals erfolgt über einen DA-Wandler mit einem R-2R-Netzwerk.

Analyse des Spektrums

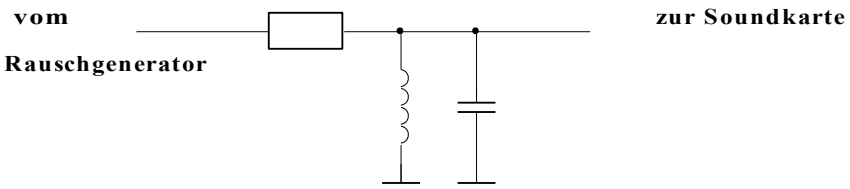
Audioanalyser ist eine Freeware die mit Hilfe der Soundkarte im PC sehr schön das Spektrum darstellen kann



Im Mittel ergibt sich ein annähernd gleichverteiltes Spektrum über den ganzen Audiobereich von 20Hz bis 20kHz. Der Abfall bei hohen Frequenzen ist wahrscheinlich durch das Anti-Aliasing Filter der Soundkarte bedingt. Bei 50Hz ist eine Brumm-Einstreuung zu sehen.

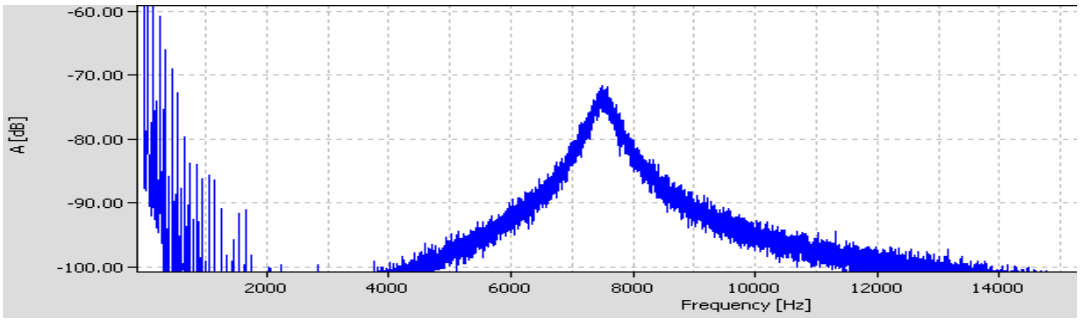
Messungen mit dem Rauschgenerator

Beispiel:



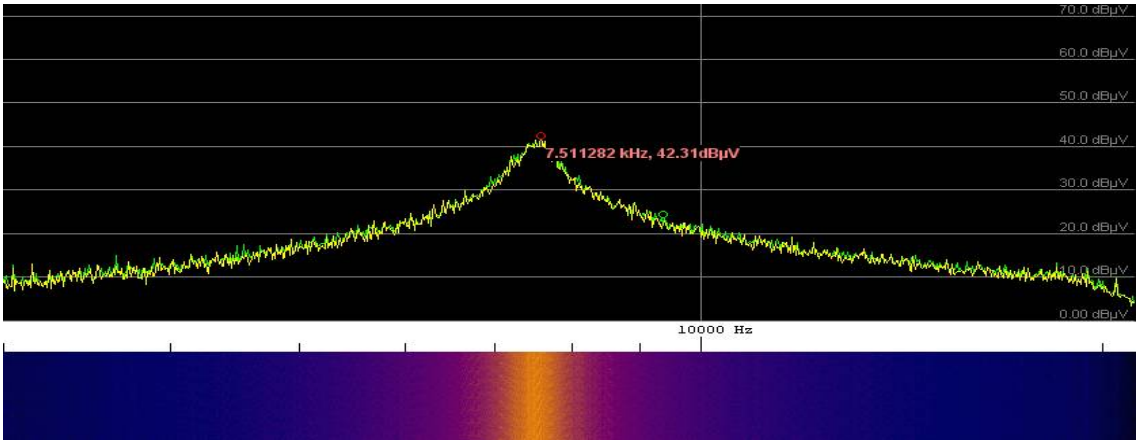
R = 100k, L=4.3mH, C = 100nF.

In Audioanalyser:



Die Resonanzfrequenz lässt sich noch recht genau bestimmen, obwohl die Spannung am Ausgang nur noch ca. 20mVss beträgt. Wegen des nicht abgeschirmten Aufbaus sind im tieffrequenten Bereich eine Menge Störsignale zu sehen.

Ein anderes Programm mit noch mehr Möglichkeiten zur Darstellung des Spektrums ist Spectrumlab Hiermit ergibt sich dieses Bild:



Anhang 1:
Andere Methoden zur Erzeugung eines Pseudo-Random-Signals

Linearer Kongruenzgenerator

Dies ist sozusagen die analoge Variante der Schieberegister-Methode

Der Mikrocontroller macht eine Iteration der Form

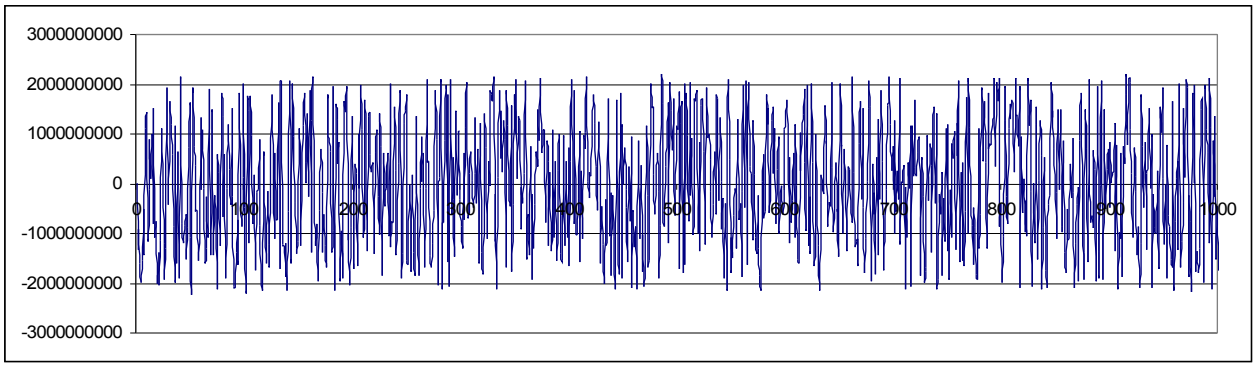
$$x_k = [a \cdot x_{k-1} + c] \, mod \, m$$

wobei a und c spezielle Startwerte haben:

		a	c
Word	16 bit	42041	1617
Long	32 bit	1664525	32767

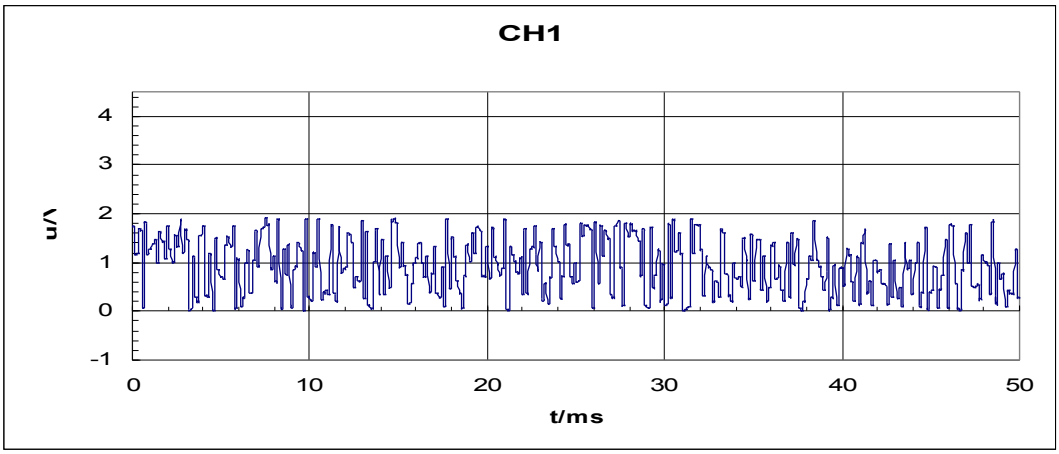
Test mit Ausgabe der Zufallswerte über die serielle Schnittstelle

```
'Linearer Kongruenzgenerator

$regfile = "m8def.dat"
$crystal = 16000000
$hwstack = 100
$swstack = 100
$framesize = 100

$baud = 9600

Dim Xold As Long
Dim X As Long
Dim Xtmp As Long


Const A = 1664525
Const C = 32767

Do
   Xtmp = A * Xold
   X = Xtmp + C
   Xold = X

   Print X

   Waitms 1
Loop
```



Mit Ausgabe über DA-Wandler:

```
'Linearer Kongruenzgenerator
'8bit   mit DA-Wandler
'High Byte benutzt

 'Spektrum bis ca. 4kHz

$regfile = "m8def.dat"
$crystal = 16000000
$hwstack = 100
$swstack = 100
$framesize = 100

'$baud = 9600

Config Portd = Output


Dim Xold As Long
Dim X As Long
Dim Xtmp As Long
'Dim Xout As Byte


'Dim I As Long


Const A = 1664525
Const C = 32767

Do
   Xtmp = A * Xold
   X = Xtmp + C
   Xold = X

   Portd = High(x)
Loop
```

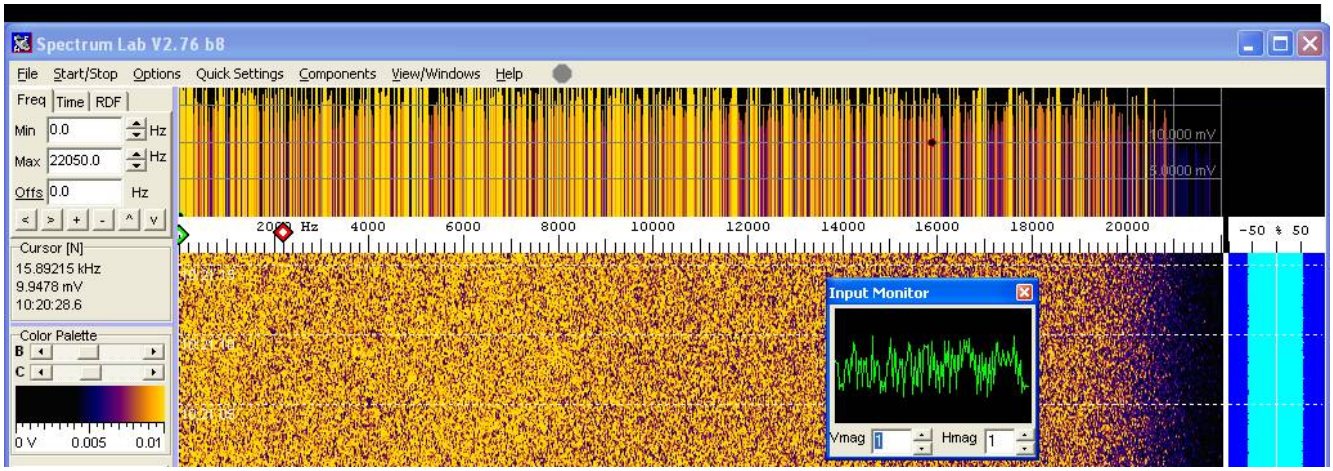


Das Spektrum reicht nur bis ca. 4kHz (mit Spektrumlab untersucht über die Soundkarte). Dabei ist es egal welches der 3 höchstwertgsten Bytes man nimmt.

Nur das niederwertigste Byte ist schlecht geeignet, da es eine periodische Änderung enthält. Die 2 untersten Bits davon sind reine symmetrische Rechtecksignale, die anderen auch sehr regelmäßig. Dies merkt man, wenn man sich die Signale der einzelnen Bits anhört.

Die oberen 24 Bit , vor allem die oberen 16, hören sich sehr verrauscht an.
Im Spektrum sieht man variierende Frequenzlinien.

In der 16-Bit-Realisierung hat man ein breiteres Spektrum (bis ca. 5 oder 6 kHz), allerdings mit abnehmender Amplitude bei hohen Frequenzen, und einer gerade im Wasserfalldiagramm wahrnehmbaren Regelmässigkeit.



Anhang 2: Analyse der Zufallswerte der LFR-Methode

Sind die Werte 0...255 für den 8-Bit-Ausgang gleichverteilt?
Um diese Frage zu beantworten wurde die Firmware leicht modifiziert und die Werte über eine serielle Schnittstelle ausgegeben.

```
'Rauschgenerator mit der Schieberegister-Methode
'32 bit
' Analyse der Byte-Werte über Terminal


$crystal = 16000000
$regfile = "m8def.dat"
    '$regfile = "2313def.dat"

Config Portb = Output

$baud = 9600
Dim X As Byte

$hwstack = 100
$swstack = 100
$framesize = 100



'-------------------------------------------------------------
    $asm
```

```
    LDI R21,0
    LDI R22,0           ;fuer XOR benutzt

    ldi R16, &h5C        ;Anfangswerte
    LDI R17, &hD3
    LDI R18, &hBC
    LDI R19, &HB4

0:
    LDI R20,32          ;4*8 = 32 bit
1:
    ROL R16             ;rotieren 3 Bytes ueber Carry
    ROL R17
    ROL R18
    ROL R19

    'Vorsicht: Bitnummerierung in der Literatur geht von 1..n
    'nicht von 0...n-1
    BST R19,7           ;Bit 32 = Bit 7 in R19 -> T-Flag
    BLD R21,0             ;dieses vom T-Flag in R21, Bit 0
    BST R18,5           ;Bit 22 = Bit 5 in R18 -> T-Flag
    BLD R22,0             ;dieses vom T-Flag in R22, Bit 0
    EOR R21,R22             ;XOR
    BST R16,2           ;Bit 2 = Bit 1  in R16 -> T-Flag
    BLD R22,0             ;dieses vom T-Flag in R22, Bit 0
    EOR R21,R22             ;XOR
    BST R16,0           ;Bit 1 = Bit 0  in R16 -> T-Flag
    BLD R22,0             ;dieses vom T-Flag in R22, Bit 0
    EOR R21,R22             ;XOR
    BST R21,0             ;R21, Bit 0 -> T-Flag
    BLD R16,0            ;T-Flag -> R16, Bit 0

    !Out PortB , R18

    sts {x},R18

    $end Asm


    Gosub Printresult

    $asm


    DEC r20             ;schon fertig?
    BRNE 1              ;wenn nicht, weiter

    RJMP 0

    $end Asm

'-----------------------------------------------------------------------
Printresult:
    push R16
    push R17
    push R18
    push R19
    push R20
    push R21
    push R22
    in r22, sreg
    push r22

    Print X
    Waitms 10


    pop r22
    !Out Sreg , R16
    pop R22
    pop R21
    pop R20
    pop R19
    pop R18
    pop R17
    pop R16
Return
```

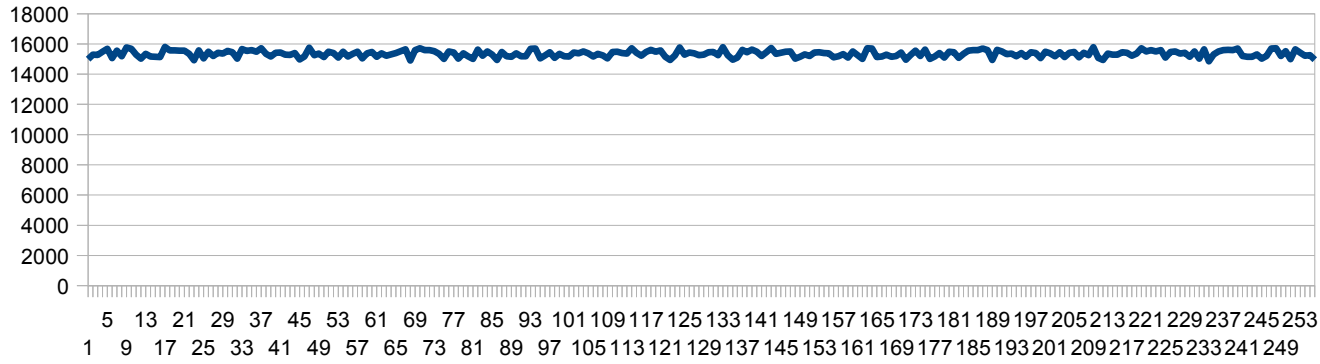
Am PC wurden die Werte dann von einem Python-Programm in Empfang genommen, in Histogramm-

Werte verrechnet und angezeigt.

Python Programm zum Speichern der Histogramm-Werte:

```
# Verteilung der Zufallswerte ermitteln.
#Die Werte kommen ueber die serielle Schnittstelle an
# Ihre Haeufigkeit wird im Array a abgespeichert

import serial
# configure the serial connections and open port
ser = serial.Serial(port='/dev/ttyS0',      baudrate=9600,
       parity=serial.PARITY_NONE,    stopbits=serial.STOPBITS_ONE,
       bytesize=serial.EIGHTBITS,    timeout=1000)
ser.open()
ser.isOpen()

#Array for histogram values
a=[0]*256
#counter , write to file only when j=...
j=0

#Main
while 1:
        #Read value from serial port and convert to integer
        sread = ser.readline()
        x=int(sread)

        #Increment the corresponding histogram value
        for i in range(0,256):
                if x==i:
                        a[i]=a[i]+1

        #show current value and all histogram values on screen
        print x
        print a

        #every now and then (if j=... ) print the values to a file
        j=j+1
        if j ==10000:
                j=0

                f=open("/home/jcf/testrandom.dat","a")
                for i in range(0,256):
                        f.write (str(a[i]))
                        f.write ("\t")
                f.write ("\n---\n")
```

Ergebnis über fast 4 Millionen Werte:



Vertikal Häufigkeit
Horizontal Byte-Wert-1