AVR-Mikrocontroller in BASCOM programmieren

Der AD-Wandler

1. Die Eigenschaften der AD-Wandler beim Mega8

(beim Mega16 / Mega32 ist es ähnlich)
Prinzip: nach dem Wägeverfahren
Auflösung: 10bit (außer ADC4, ADC5)
Linearität: 0.5bit, Genauigkait ± 2 bit
Dauer der Wandlung:: 65...260µs beim ATMega8, abhängig von der Auflösung.
Volle Auflösung bei 50...200kHz ADC-Takt
Eine Konversion dauert 13 AD-Wandler-Taktzyklen, außer beim ersten Mal, dann sind es 25.
Eine interne Referenzspannung von 2.56V steht zur Verfügung

Achtung:

- ADC4, ADC5 beim AtMega8 haben nur 8 bit Auflösung!
- AVCC darf sich um max. 0.3V von VCC unterscheiden (Schottky-Diode intern)
  Am besten AVCC – Versorgung aus VCC über Tiefpass 10µH / 100nF
- AREF mit 100nF abblocken
- Die Signalquelle sollte <10kΩ Impedanz haben

Berechnung des Spannungswertes:

$$U_{ADC} = \frac{U_{REF} \cdot ADCvalue}{1024} \qquad \text{bei 10 Bit}$$

Der Wert $1024 = 2^{10}$ ergibt sich aus der Anzahl möglicher Spannungsschritte.

Die interne Referenzspannung U_{REF} hat 2.56V, leichte Abweichungen sind möglich (laut Datenblatt von 2.3V bis 2.7V beim Mega8). Um dies auszugleichen, kann man Kalibrierfaktoren einführen, siehe später.

2. Einlesen von Kanal 0, mit Ausgabe über die serielle Schnittstelle

```
'ADC_1: Kanal 0 lesen und über die serielle Schnittstelle ausgeben
$crystal = 4000000
$regfile = "m8def.dat"

'Serielle Schnittstelle
$baud = 9600

'ADC
Config Adc = Single , Prescaler = Auto , Reference = Internal
Start Adc
Dim Wadc As Word
Dim Value As Single


'-----------------------------------------------------------------
Do
    Gosub Measure
    Gosub Printvalues
    Waitms 1000
```

```
Loop
'--------------------------------------------------------------------
Measure:
    Wadc = Getadc(0)
    Value = Wadc * 2.56
    Value = Value / 1024
Return
'--------------------------------------------------------------------
Printvalues:
    Print Wadc;
    Print Chr(9);
    Print Value;
    Print "V"
Return
```

Auf die gleiche Weise können die anderen Kanäle gelesen werden.
Hier wurde die interne Referenz benutzt, damit ist die Auflösung 2.56V/1024 = ca. 2.5mV.

3. ADC mit Kalibrierfaktoren für genaueres Ergebnis

```
'ADC0 lesen und über serielle Schnittstelle ausgebn,
'mit Kalibrierfaktoren
$crystal = 4000000
$regfile = "m8def.dat"

'Serielle Schnittstelle
$baud = 9600

'ADC
Config Adc = Single , Prescaler = Auto , Reference = Internal
Start Adc
Dim Wadc As Word
Dim Value As Single
Dim Adkorr1 As Single
Dim Adkorr2 As Single
Adkorr1 = 2500 / 2517
Adkorr2 = 0

'--------------------------------------------------------------------
Do
    Gosub Measure
    Gosub Printvalues
    Waitms 1000

Loop
'--------------------------------------------------------------------
Measure:
    Wadc = Getadc(0)
    Value = Wadc * 2.56
    Value = Value / 1023
    Value = Value * Adkorr1
    Value = Value + Adkorr2
Return
'--------------------------------------------------------------------
Printvalues:
    Print Wadc;
    Print Chr(9);
    Print Value;
    Print "V"
Return
```

Hier wird das Ergebnis nach einer linearen Funktion berechnet.
Zunächst wird der ausgegebene Wert für U = 0 bestimmt und wenn nötig durch geeignete Wahl von Adkorr2 auf null gebracht.
Dann wird bei einer vollen Aussteuerung (ca. 2.5V) und Adkorr=1der Ausgabewert ermittelt und Adkorr1 bestimmt. Im Beispiel war der angezeigte Wert 2.517V bei 2.5V am Eingang.

4. Messung im Sekundentakt mit Timer – Interrupt

Die obigen Beispiele liefern Werte in einem ungenauen Zeitraster, denn erstens ist die Waitms – Funktion eine ungenaue Funktion, und zweitens braucht der Code zur Verarbeitung auch seine Zeit.

Besser ist es hier, mit Timer-Interrupt zu arbeiten.

```
$crystal = 4000000
$regfile = "m8def.dat"

'Serielle Schnittstelle
$baud = 9600

'ADC
Config Adc = Single , Prescaler = Auto , Reference = Internal
Start Adc
Dim Wadc As Word
Dim Value As Single
Dim Adkorr1 As Single
Dim Adkorr2 As Single
Adkorr1 = 2500 / 2517
Adkorr2 = 0

'Timer1 Compare -> jede Sekunde Interrupt
Dim Newtime As Bit
Dim Seconds As Long
Compare1a = 15625
Config Timer1 = Timer , Prescale = 256 , Clear Timer = 1 , Compare A = Disconnect
On Compare1a Timer1_isr
Enable Compare1a
Enable Interrupts

'-------------------------------------------------------------------
Do
  If Newtime = 1 Then
    Newtime = 0
    Gosub Getmeasure
    Gosub Printvalues
  End If
Loop
'-------------------------------------------------------------------
Getmeasure:
    Value = Wadc * 2.56
    Value = Value / 1023
    Value = Value * Adkorr1
    Value = Value + Adkorr2
Return
'-----------------------------------------------------------------------------
Printvalues:
    Print Seconds;
    Print Chr(9);
    Print Wadc;
    Print Chr(9);
    Print Value;
    Print "V"
Return
'-----------------------------------------------------------------------------
Timer1_isr:
      Wadc = Getadc(0)
      Incr Seconds
      Newtime = 1
Return
```

Bei jedem Interrupt wird ein Wert vom AD-Wandler gemessen und das Flag Newtime gesetzt. Die Verarbeitung (Verrechnung) mit dem Unterprogramm Getmeasure erfolgt erst durch einen Aufruf aus dem Hauptprogramm, damit die Interruptroutine so kurz wie möglich gehalten werden kann.

Das zeitliche Verhalten kann sehr schön im Simulator untersucht werden.
In diesem Beispiel läuft der ADC im SINGLE Modus, eine Messung wird erst gestartet wenn der Timer-Interrupt aufgerufen wird. Die Latenzzeit bis der Wert in der Variablen

Wadc steht ist hier 80 Taktzyklen, also 20µs bei 4MHz.

Davon entfallen allein 53 Takte auf das Pushen der Register bevor wir in der eigentlichen Interrupt-Routine landen. Dieser Teil könnte verkürzt werden, wenn mit der Option NOSAVE gearbeitet wird und man sich selbst um das Pushen nur der benutzten Register kümmert. (siehe BASCOM und Assembler).

Diese Latenzzeit stört in unserem Fall nicht, sie muss aber verkürzt werden wenn man sehr schnell abtasten will.

Eine Möglichkeit dafür ist es, den AD-Wandler frei laufen zu lassen und beim Timer-Interrupt nur das ADC – Register zu lesen.

Bei der Konfiguration wird folgendes geändert:

```
'ADC
Config Adc = Free , Prescaler = Auto , Reference = Internal
Start Adc
Dim Wadc As Word
Dim Wadcl As Byte At Wadc Overlay
Dim Wadch As Byte At Wadc + 1 Overlay
```

Mit ADC = FREE läuft der AD-Wandler dauernd.
Die Variablen Wadcl und Wadcl liegen wegen Overlay an der Adresse Wadc, sodass Low- und High- Byte beim Lesen automatisch an der richtigen Stelle landen.

In der Timer- Interruptroutine wird nur das ADC – Register in 2 Byte – Häppchen gelesen:

```
Timer1_isr:
      'Wadc = Getadc(0)
      Wadcl = Adcl
      Wadch = Adch
      Incr Seconds
      Newtime = 1
Return
```

Diese Änderung bringt die Latenzzeit von 80 auf 63 Zyklen herunter.

Wo wird in diesem Beispiel eigentlich der AD-Kanal ausgewählt? Gar nicht, bzw. implizit dadurch, dass der Config- Befehl die Bits 0...3 des ADMUX-Registers auf 0 setzt. So wird automatisch Kanal 0 ausgewählt. Will man einen anderen Kanal haben, muss man diese Bits von Hand setzen, zB. Für Kanal 1:

```
'ADC
Config Adc = Free , Prescaler = Auto , Reference = Internal
Admux.0 = 1
Admux.1 = 0
Admux.2 = 0
Admux.3 = 0
Start Adc
```

5. Ein Blick auf die Register

Es gibt mindestens zwei Gründe, sich einmal näher mit der Hardware des Controllers zu beschäftigen:

- um die korrekte Kompilierung von BASCOM zu überprüfen
- um etwas zu tun, was BASCOM (noch) nicht erlaubt. Nicht alle verfügbaren Optionen der Controller sind schon in BASCOM integriert.

Zur Überprüfung der Kompilierung eignet sich sehr gut der Simulator. Mit ihm lassen sich Registerinhalte und Zeitverhalten sehr gut überprüfen.

ADMUX:

7	6	5	4	3	2	1	0
REFS1	REFS0	ADLAR	MUX4*)	MUX3	MUX2	MUX1	MUX0

*) 0 beim Mega8, MUX4 beim Mega16

REFS1	REFS0	Referenzspannung
0	0	Extern an AREF
0	1	AVCC, mit Kondensator an AREF
1	0	-
1	1	Intern, 2.56V, mit Kondensator an AREF

ADLAR	
0	Rechtsbündige Darstellung des Wertes (Wert in Bits 0...9)
1	Linksbündige Darstellung des Wertes (Wert in Bits 6...15)

MUX4...0	
00000 00001 00010 ... 00111	ADC0 ADC1 ADC2 ... ADC7
01000 ... 11101	Nur beim Mega16/32, nicht beim Mega8: Differentielle Modes und verstärkte Modes siehe Datenblatt!
11110	Interne Bandgap – Referenz 1.23V (für Kalibrierzwecke)
11111	0V (für Kalibrierzwecke)

ADCSRA (ADC Control and Status Register):

7	6	5	4	3	2	1	0
ADEN	ADSC	ADFR *) ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

Achtung: dieses Register kann in unterschiedlichen .def-Dateien auch ADCSR heissen!

ADEN = 1: ADC Enable
ADSC = 1: Start Conversion
ADFR = 1: Free running mode selected (beim Mega8)
ADATE = 1: Auto Trigger Enable (beim Mega16/32). Die Triggerquelle wird im SFIOR-Register ausgewählt.
ADIF: AD-Wandler-Interrupt flag. Wird gesetzt wenn die Wandlung abgeschlossen ist.
Löst den Interrupt aus wenn ADIE = 1. (Flag wird automatisch rückgesetzt)
ADIE = 1: AD Interrupt enable

ADP: Prescaler

ADP2...0	Prescale- Faktor
000	2
001	2
010	4
011	8
100	16
101	32
110	64
111	128

Der Prescale – Faktor legt den Teilerfaktor für den AD-Wandlertakt fest, z.B. ergibt sich bei einem 4MHz Quarz und ADP = 101 eine Ferquenz von 125kHz.
Bei voller Auflösung (10 Bit) sollte die Taktfrequenz des AD-Wandlers zwischen 50 und 200kHz liegen, bei 8 Bit Auflösung kann man bis zu 1MHz gehen.

Wenn man diese Register-Bedeutungen kennt, kann man auch, manchmal bequemer als mit CONFIG..., die gewünschte Konfiguration bekommen. Allerdings muss man beim Wechsel des Controllers aufpassen ob sich nicht die Registerbelegung geändert hat (z.B. sind beim Mega8 und Mega16 sehr viele Ähnlichkeiten, aber auch einige Differenzen in der Registerbelegung).

Beispiel:

```
'lesen Kanal 0 mit interner Referenz, nur 8 Bit
'Byte-Werte werden direkt über RS232 geschickt
$regfile = "m8def.dat"
$crystal = 4000000
$baud = 9600


Config Portc = Input
Admux = &B11100000                      'REFS=11 interne Referenz,
                                        'ADLAR=1: linksbündig, höchstwertigste 8 bit in ADCH
                                        'MUX = 0 -> ADC0

Adcsra = &B11100010                     '1 ADEN, 1 Start, 1 Free run,
                                        '00 IntFlag = 0, kein Interrupt
                                        '010 Prescaler = 4  -> 1MHz Takt
Do
   Print Adch
   Waitms 2
Loop
```

In diesem Beispiel ergibt sich natürlich keine genau zeitlich äquidistante Abtastung, denn erstens ist der Waitms – Befehl relativ ungenau und zweitens läuft der ADC frei, so dass der Moment der Abtastung nicht genau definiert ist.

6. ADC – Werte mit genauem Zeitraster

Beim Mega16 / Mega32 kann die Triggerung des ADC hardwaremäßig direkt durch einen Timer – Interrupt erfolgen (Auto Trigger). Dies hat den Vorteil dass die Zeitabstände genau stimmen, da keine softwaremäßige Latenzzeit aftritt. Beim Mega8 ist dies leider nicht möglich.

BASCOM unterstützt diese Option nicht direkt, es bleibt einem also nichts anderes übrig als sich auf die Datenblätter zu stürzen und die Register von Hand zu setzen.

Für den Mega32 muss folgendes getan werden:

1. Setzen des ADMUX-Registers zur Auswahl des Kanals, der Referenz (intern / extern) und der Darstellung des Wertes (links- oder rechtsbündig)
   (Diese Einstellung könnte auch mit einem Config-Befehl erfolgen).

2. Setzen von ADATE im ADCSRA-Register, so dass die Autotrigger-Funktion aktiviert ist.
   In diesem Register wird auch der Prescale-Wert für den Takt eingestellt.

3. Wahl der Triggerquelle, hier Timer1-Overflow oder Timer1- Compare B – Ereignis
   durch die Bits 7,6,5 im SFIOR-Register.
   Wenn dies geschehen ist, wird der ADC hardwaremäßig durch den Timer ausgelöst, ohne dass man sich weiter darum kümmern muss.

4. Einrichten eines ADC Complete Interrupts.
   Wenn eine AD-Wandlung abgeschlossen ist, wird dieser Interrupt aufgerufen.
   In der Interruptroutine muss dann der Wert abgespeichert werden.
   Ein Flag kann dem Hauptprogramm mitteilen, dass der Wert zur Verfügung steht.

5. Konfiguration von Timer1 für ein Compare B – Ereignis im gewünschten Zeitraster.
   Dadurch wird automatisch schon der ADC getriggert.
   Eventuell konfiguriert man aber zusätzlich einen Timer1-Interrupt, um eine Zeitvariable zu inkrementieren oder sonstwas im gleichen Zeitraster zu erledigen.

Im Einzelnen:

1. Setzen von ADMUX:

```
Admux = &B11000000                    'REF internal, result in bits 0...9, channel 0
```

2. Setzen von ADCSRA:

Aktivieren der Autotrigger – Funktion aktiviert werden: ADATE = 1 (Auto Trigger Enable)

```
Adcsra = &B11101111                   'ADC Enable, Start Conversion, Auto trigger enable,
                                       'ADIF = 0
                                       'AD int enable, 111 = Prescale 128
```

ADCSRA (ADC Control and Status Register):

7	6	5	4	3	2	1	0
ADEN	ADSC	ADFR *) ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

Für die restlichen Bits gilt:
ADEN = 1 (ADC Enable)
ADSC = 0 (Start Conversion = 0)
ADIF = 0 (Interrupt flag = 0)
ADIE = 1 (ADC interrupt enable)

Mit ADPS wird der Prescaler eingestellt, z.B. 111 entspricht 128, das ergibt bei 16MHz Taktfrequenz 16000kHz/128 = 125kHz Takt für den AD-Wandler, was OK ist, denn er soll zwischen 50 und 200kHz liegen.

3. Auswahl der Triggerquelle im SFIOR – Register

```
Set Sfior.7                          '101 =  Timer1 Match B
Reset Sfior.6
Set Sfior.5
```

SFIOR

7	6	5	4	3	2	1	0
ADTS2	ADTS1	ADTS0	-	ACME	PUD	PSR2	PSR10

ADTS2...ADTS0	Triggerquelle
000	Free running mode
001	Analog Komparator
010	Ext. Interrupt 0
011	Timer0 Compare Match
100	Timer0 Overflow
101	Timer Compare Match B
110	Timer1 Overflow
111	Timer 1 Capture Event

4. ADC Complete Interrupt

```
On Adcc Adcinterrupt                 'ADCC = AD Conversion Complete
....
....
'-------------------------------------------------------------------------------
Adcinterrupt:
'is executed when ADC is ready
'updates Wadc and sets flag Adc_complete
      Wadcl = Adcl
      Wadch = Adch
      Adc_complete = 1
Return
```

Die Bytes ADCL und ADCH werden in die Variablen WADCL und WADCH gespeichert. Wenn diese mit der Overlay-Option an der gleichen Speicheradresse wie WADC definiert sind,

```
Dim Wadc As Word
Dim Wadcl As Byte At Wadc Overlay
Dim Wadch As Byte At Wadc + 1 Overlay
```

braucht man keine Umwandlung, das Ergebnis steht als Word gleich inder Variablen WADC.

In der ADC-Interruptroutine wird noch das Flag Adc_complete gesetzt.
Das Hauptprogramm kann dieses abfragen und so herausfinden ob ein neuer Wert zur Verfügung steht.

5. Konfiguration des Timer1 für ein Compare B – Ereignis.

```
Compare1a = 15625
Config Timer1 = Timer , Prescale = 1024 , Clear Timer = 1 , Compare B = Disconnect
Enable Compare1b
```

Hier muss man ein wenig aufpassen: obwohl ein Compare B benutzt wird (dies muss man da die Autotrigger-Funktion Compare A nicht erlaubt), muss der Compare-Wert selbst in das Register COMPARE1A geschrieben werden.

Hier ein vollständiges Programm welches jede Sekunde einen Wert misst:

```
'Read ADC(0) every second and output it via RS232
'ADC is triggered automatically by timer1 Compare match
$crystal = 16000000
$regfile = "m32def.dat"

'Serielle Schnittstelle
$baud = 9600

'.....................................................................
'ADC

 Admux = &B11000000                    'REF internal, result in bits 0...9, channel 0
 Adcsra = &B11101111                   'ADC Enable, Start Conversion, Auto trigger enable,
                                       'ADIF = 0
                                       'AD int enable, 111 = Prescale 128

 Set Sfior.7                           '101 =  Timer1 Match B
 Reset Sfior.6
 Set Sfior.5

 On Adcc Adcinterrupt                  'ADCC = AD Conversion Complete

Dim Wadc As Word
Dim Wadcl As Byte At Wadc Overlay
Dim Wadch As Byte At Wadc + 1 Overlay
Dim Value As Single
Dim Adkorr1 As Single
Dim Adkorr2 As Single
Dim Adc_complete As Bit
Adkorr1 = 2500 / 2560
Adkorr2 = 0
'.....................................................................

'Timer1 Compare -> interrupt every second
'Compare B is used as it must be used for ADC auto trigger
'But: altogh Compare B, the compare value must be in Compare1A!
Dim Seconds As Long
Compare1a = 15625                      'COMPARE1A wird benutzt, obwohl COMPARE B Ereignis
auslöst!
Config Timer1 = Timer , Prescale = 1024 , Clear Timer = 1 , Compare B = Disconnect
On Compare1b Timer1_isr
Enable Compare1b
'.....................................................................
Enable Interrupts

'-------------------------------------------------------------
Do
  'when ADC is ready, flag Adc_complete is 1
  'then get and print value
  If Adc_complete = 1 Then
    Adc_complete = 0
    Gosub Getvalue
    Gosub Printvalues
  End If

Loop
'-------------------------------------------------------------
Getvalue:
    'Wadc is updated by ADC complete interrupt
    'so we only have to get the value
    Value = Wadc * 2.56
    Value = Value / 1023
    Value = Value * Adkorr1
    Value = Value + Adkorr2
```

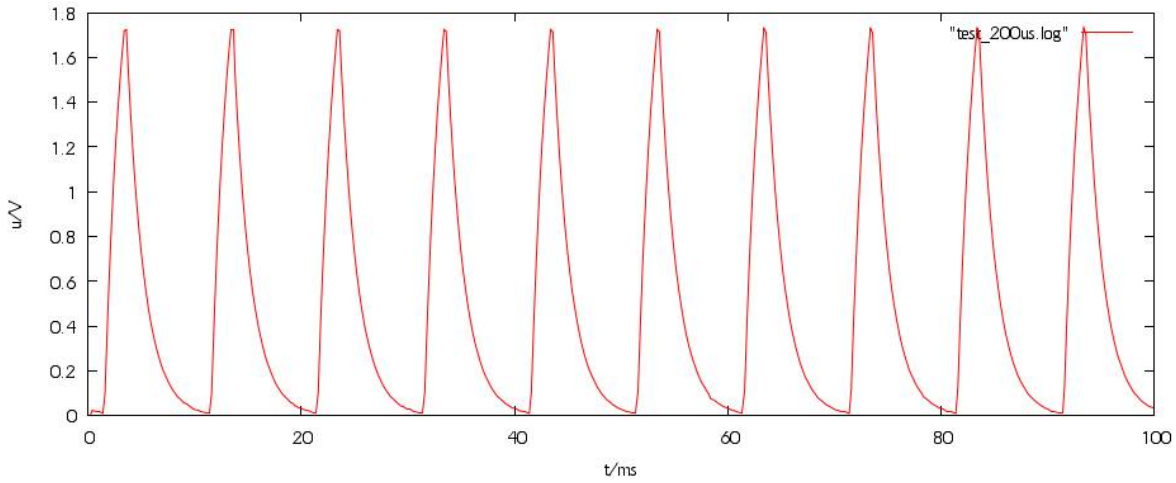
```
Return
'--------------------------------------------------------------------------------
Printvalues:
    Print Seconds;
    Print Chr(9);
    Print Wadc;
    Print Chr(9);
    Print Value;
    Print "V"
Return
'--------------------------------------------------------------------------------
Timer1_isr:
'increments Seconds
'ADC is automatically triggered by hardware, as set by  intitialization
' in registers ADCSRA and SFIOR
      Incr Seconds

Return
'--------------------------------------------------------------------------------
Adcinterrupt:
'is executed when ADC is ready
'updates Wadc and sets flag Adc_complete
      Wadcl = Adcl
      Wadch = Adch
      Adc_complete = 1
Return
```

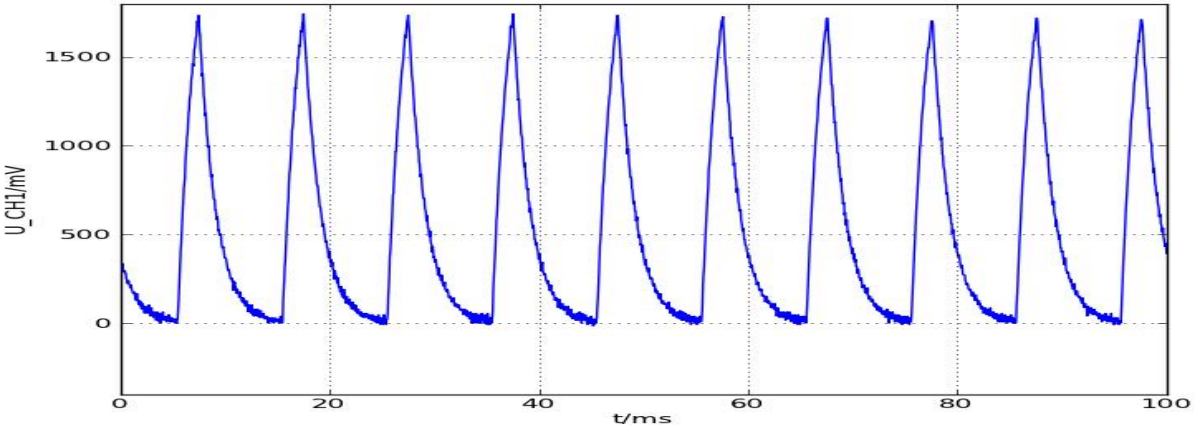
7. Schnelles Abtasten

Wenn es schnell gehen soll, z.B. wenn man den Controller als Digital-Oszilloskop nutzen will, bleibt keine Zeit, die Daten sofort auszugeben. Sie müssen in einem Array zwischengespeichert werden. Die Übertragung erfolgt dann erst wenn das Einlesen beendet ist.

Das folgende „Oszillogramm“ wurde auf diese Weise bei einer Auflösung von 10 Bit und einem Abtastintervall von 200µs erzeugt. Die 500 Datenpunkte wurden seriell übertragen, in einem Terminalprogramm geloggt und mit Gnuplot gezeichnet.
Als Signalquelle diente ein TTL-Signal von 100Hz, welches über einen Tiefpass-Spannungsteiler 10k / 10k//330nF verschliffen wurde.



Zum Vergleich eine Messung mit dem Digitalspeicher-Oszilloskop:



Auffallend ist, dass die Messung mit dem Mikrocontroller weniger verrauscht ist, da die Messung mit Controller 10Bit Genauigkeit hat, während das Oszilloskop nur mit 8 Bit arbeitet.

Hier das Programm dazu:

```
'Read ADC(0) every 200us and output t, U via RS232
'ADC is triggered automatically by timer1 Compare match
$crystal = 16000000
$regfile = "m32def.dat"

'Serielle Schnittstelle
$baud = 9600

'.....................................................................
'ADC

 Admux = &B11000000                    'REF internal, result in bits 0...9, channel 0
 Adcsra = &B11101111                   'ADC Enable, Start Conversion, Auto trigger enable,
                                       'ADIF = 0
                                       'AD int enable, 111 = Prescale 128

 Set Sfior.7                           '101 =  Timer1 Match B
 Reset Sfior.6
 Set Sfior.5

 On Adcc Adcinterrupt                  'ADCC = AD Conversion Complete

Dim W(500) As Word                     'save 500 values
Dim Wadc As Word
Dim Wadcl As Byte At Wadc Overlay
Dim Wadch As Byte At Wadc + 1 Overlay
Dim Value As Single
Dim Adkorr1 As Single
Dim Adkorr2 As Single
Dim Adc_complete As Bit
Adkorr1 = 2500 / 2560
Adkorr2 = 0
Dim I As Long
Dim T As Single

'.....................................................................

'Timer1 Compare -> interrupt every 200us
'Compare B is used as it must be used for ADC auto trigger
'But: altogh Compare B, the compare value must be in Compare1A!
Dim Seconds As Long
Compare1a = 3200                       'Compare value MUST be in COMPARE1A !
Config Timer1 = Timer , Prescale = 1 , Clear Timer = 1 , Compare B = Disconnect
'On Compare1b Timer1_isr
Enable Compare1b
'.....................................................................
Enable Interrupts

'-----------------------------------------------------------
```

```
'MAIN

I = 1

'Read 500 values every 100us
'Done automatically by Timer1 Compare B -> ADC
Do
Loop Until Adc_complete = 1

'Interrupts no longer needed
Disable Interrupts

'Output all values via RS232
For I = 1 To 500
   Wadc = W(i)
   Gosub Calculatevalue
   Gosub Printvalue
   Waitms 5
Next I

Do
Loop
'-----------------------------------------------------------------
Calculatevalue:

    Value = Wadc * 2.56
    Value = Value / 1023
    Value = Value * Adkorr1
    Value = Value + Adkorr2
Return
'-------------------------------------------------------------------------------
Printvalue:
   T = I * 0.2                          'time in ms
   Print T ;
   Print Chr(9);
   Print Value                          'voltage
Return
'-------------------------------------------------------------------------------
Adcinterrupt:
'is executed when ADC is ready
'updates Wadc and sets flag Adc_complete
      Incr I
      Wadcl = Adcl
      Wadch = Adch
      W(i) = Wadc
      If I = 500 Then Adc_complete = 1

Return
```

Bei einer Abtastzeit von 100µs funktioniert dieses Programm nicht mehr korrekt. Eine Untersuchung mit dem Simulator zeigt, dass nicht der Zeitverbrauch des Adcinterrupts das Problem ist, sondern die Zeit die der Ad-Wandler braucht zum Wandeln. Eine Umstellung des ADC-Prescalers von 128 auf 64 erhöht die ADC-Taktfrequenz von 125 auf 250kHz (was eigentlich schon zuviel ist), und das Programm funktioniert wieder.

Will man noch schneller abtasten, kann man die Frequenz bis auf 1MHz erhöhen und nur eine Auflösung von 8 bit benutzen.

Ein interessanter Artikel von B. Kainka zum Thema Oszilloskop mit Mikrocontroller findet sich in Elektor 3.2009