# AD-Umsetzung mit megaAVR

Steuerung des internen AD-Umsetzers mit BASCOM-AVR

**Dr. Claus Kühnel**

Die Familie der AVR RISC Mikrocontroller von Atmel wächst am unteren Ende mit den tinyAVRs und am oberen Ende mit den megaAVRs ständig.

Für die Messwerterfassung von besonderem Interesse sind neben der üblichen Peripherie die bei vielen AVR Mikrocontrollern auf dem Chip vorhandenen 10-Bit AD-Umsetzer. Abbildung 1 zeigt ein Blockschema eines AVR Mikrocontrollers. Der Teil, der uns im weiteren beschäftigen wird, ist markiert.

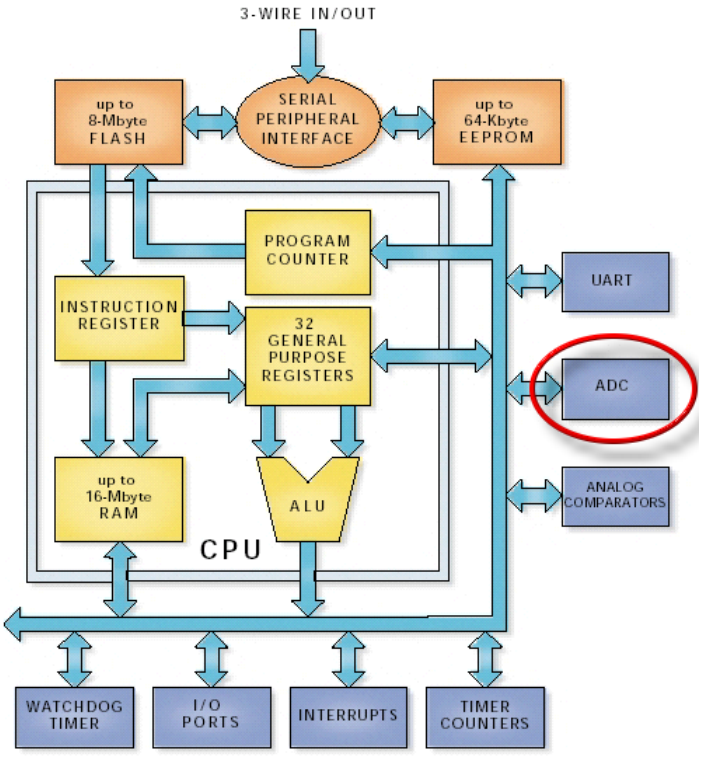


**Abbildung 1 Blockschema AVR Mikrocontroller**

# 1. ATmega32

Der ATmega32 dient in diesem Beitrag als Hardwarebasis, da es geeignete Mikrocontrollermodule unterschiedlicher Hersteller für den Aufbau von Prototypen gibt.

Neben umfangreicher Peripherie kann der ATmega32 auch einen achtkanaligen 10-Bit AD-Umsetzer aufweisen. Angeboten wird der ATmega32 im 44-poligen TQFP oder 40-poligen PDIP Gehäuse. Abbildung 2 zeigt die Pinbelegung des ATmega32 im TQFP Gehäuse einschließlich der alternativen Pinfunktionen.

Wer schon mal mit AVR Mikrocontrollern zu tun hatte, kann hier leicht auf die weitere, zur Verfügung stehende Peripherie schließen.

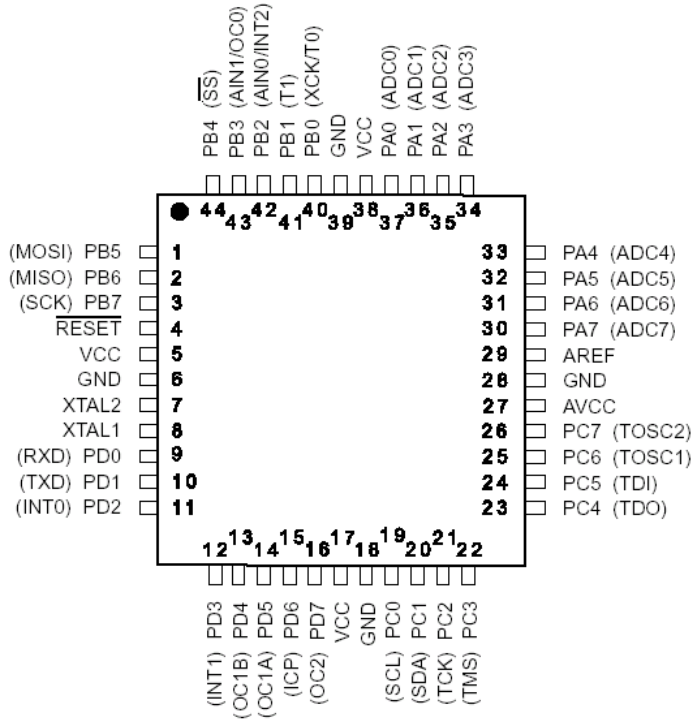


**Abbildung 2 ATmega32 Pinbelegung TQFP**

Entscheidet man sich z.B. aus Platzgründen für das in Abbildung 2 gezeigte TQFP Gehäuse, dann ist man zumindest für den Prototypenbau oder ein erstes Kennenlernen gut mit einem Mikrocontrollermodul beraten.

# 2. stAVeR-24M32 Mikrocontrollermodul

Zur Untersuchung des AD-Umsetzers im ATmega32 kommt hier das Mikrocontrollermodul stAVeR-24M32 der schwedischen Firma Lawicel [www.lawicel.com] zum Einsatz, was in Deutschland über den Elektronikladen in Detmold [www.elektronikladen.de] bezogen werden kann.

Die Eigenschaften des StAVeR-24M32 Mikrocontrollermoduls sind weitgehende durch den verwendeten Mikrocontroller ATmega32 festgelegt. StAVeR-24M32 weist folgende Merkmale auf:

- 32 KByte Flash Memory (30 KByte für Code, 2 KByte für Flash Boot Code)
- 1 KByte EEPROM
- 2 KByte SRAM
- 14.7456 MHz Quarz On-Board
- 16 I/O Leitungen an den DIL Anschlüssen
- 3 I/O Leitungen (SPI Bus) am (roten) ISP Anschluss
- ein externer Interrupt (INT1)
- zwei 8-Bit Timer
- ein 16-Bit Timer
- 8-Kanal 10-Bit AD-Umsetzer
- $I^2C$ Hardware Interface
- SPI Interface (Master) am (roten) ISP Anschluss
- programmierbar über RS-232 Programmdownload (keine Notwendigkeit für externen Programmer)
- ESD geschütztes RS-232 Interface mit bis zu 115.2 kBaud
- erweiterter Temperaturbereich – 40 °C bis + 85 °C
- 3 LEDs (grün, gelb, rot) On-Board (über Software ansteuerbar).

Abbildung 3 zeigt das kompakte Modul.

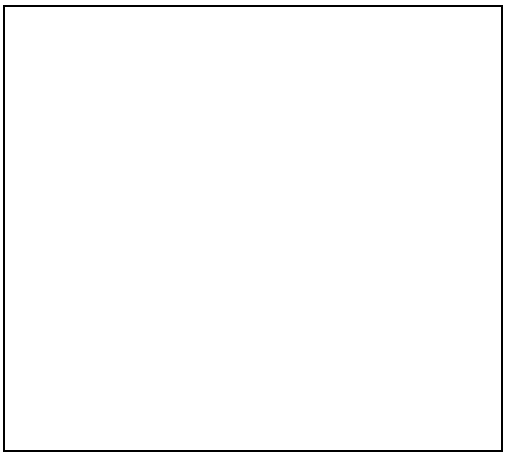

**Abbildung 3 stAVeR-24M32 Mikrocontrollermodul**

Wenn man außerdem bedenkt, dass beim Löten eines 44-poligen TQFP u.U. wenig Freude aufkommt, dann ist man schnell für den Einsatz eines solchen Mikrocontrollermoduls

motiviert. Zusätzlich zum stAVeR-24M32 kann man sich auch das passende Evaluationboard vom gleichen Hersteller beschaffen und hat damit eine lauffähige Infrastruktur beisammen.

Wer sich einen Überblick über mögliche Alternativen schaffen will, wird auf der Website [http://www.avrfreaks.net/Tools/jindex.php?preopen=Development+Board] gut beraten.

Die megaAVR bieten neben der ISP-Programmierung die Möglichkeit für ein Programmdownload über die serielle Schnittstelle (UART). Ein Bootloader sorgt in diesen Fällen für die Programmierung des für die Anwendung benutzten Flash Memories.

Der stAVeR-24M32 ist bei Auslieferung mit einem solchen Bootloader versehen, weshalb ich von dieser Möglichkeit der Einfachheit halber Gebrauch gemacht habe. Auf einen Programmer kann so vollkommen verzichtet werden.

Beim stAVeR-24M32 sind 24 Anschlüsse vorhanden. Nicht alle Pins des ATmega32 können deshalb nach außen geführt sein. Abbildung 4 zeigt die Pinbelegung beim stAVeR-24M32.

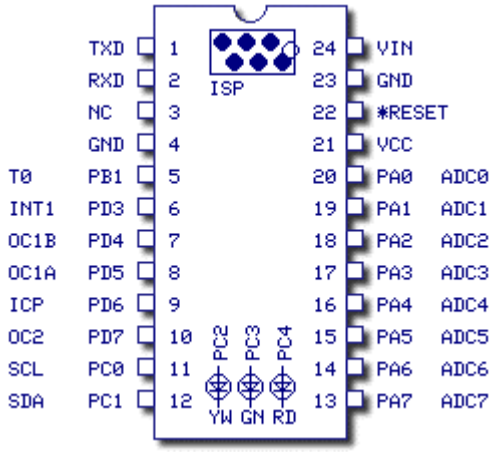


**Abbildung 4 stAVeR-24M32 Pinbelegung**

Die für uns wichtigen Anschlüsse des AD-Umsetzers stehen zur Verfügung. Wie leicht zu erkennen ist, kann das 24-polige Modul in jeden Standard-EPROM-Sockel gesteckt werden.

Wenn weitere Anschlüsse benötigt werden, dann kann man z.B. auf den stAVeR-40 umsteigen, der zusätzliche Peripherie, wie RTC, $I^2C$-EEPROM und einen 8-Bit DA-Umsetzer, auf dem Board zur Verfügung stellt. Abbildung 5 zeigt das stAVeR-40 Mikrocontrollermodul.

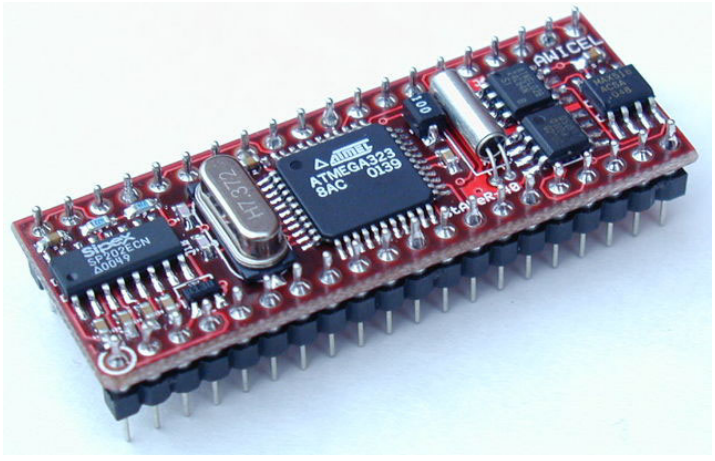

**Abbildung 5 stAVeR-40 Mikrocontrollermodul**

# 3. Interner AD-Umsetzer

Der interne AD-Umsetzer der in vielen megaAVR und einigen tinyAVR heute zu finden ist, weist die folgenden Merkmale auf:

- Auflösung 10 Bit
- integrale Nichtlinearität 0.5 LSB, absolute Genauigkeit ± 2 LSB
- Umsetzzeit von 65 bis 260 µs
- Umsetzrate von bis zu 15000 Umsetzungen pro Sekunde
- acht massebezogene Analogeingänge (single ended)
- sieben differentielle Analogeingänge
- Ergebnis der AD-Umsetzung rechts- oder linksbündig
- Eingangsspannungsbereich 0 bis AVCC
- interne Referenzspannung von 2,56 V
- kontinuierliche (free running) oder Einzelumsetzung (single shot)
- interruptgesteuerte Auslösung der AD-Umsetzung durch Auto-Triggerung
- Interrupt am Ende der AD-Umsetzung
- Rausch-Unterdrückung

Abbildung 6 zeigt einen Ausschnitt aus dem Blockschaltbild des AD-Umsetzers, welcher erkennen lässt, dass der AD-Umsetzer nach dem Verfahren der sukzessiven Approximation arbeitet.

Beim AD-Umsetzer nach dem Verfahren der sukzessiven Approximation wird die analoge Eingangsspannung mit der Ausgangsspannung eines DA-Umsetzers verglichen. Die Ausgangsspannung des DA-Umsetzers wird durch die Steuerlogik sowie eine der Referenzspannungen festgelegt. Eine Steuerlogik steuert den DA-Umsetzer bitweise an, und das Komparatorausgangssignal bestimmt, ob das jeweilige Bit im Ausgaberegister gesetzt oder nicht gesetzt wird. Auf diese Weise nähert sich die Ausgangsspannung des DA-Umsetzers sukzessive dem zu erfassenden analogen Spannungswert.

Dieser Spannungswert darf sich während des Umsetzvorgangs nicht ändern, da sonst ein falscher Inhalt des Ausgaberegisters die Folge wäre. Eine dem Komparator vorgeschaltete Sample&Hold- Schaltung erfüllt hier diese Forderung.

Die erforderliche Umsetzzeit ist unabhängig von der anliegenden Eingangsspannung und richtet sich nur nach der Auflösung des AD-Umsetzers. Ein 10-Bit AD-Umsetzer benötigt genau zehn Umsetzschritte, deren Zeit durch die Taktung des DA-Umsetzers und die Schaltzeit des Komparators bestimmt wird.

Die Eingangsspannung wird über zwei Multiplexer an den Komparator geführt. Für Kalibrationszwecke können zusätzlich die interne Bandgap-Referenzspannung und das Massepotential an den Komparator geführt werden. Ein programmierbarer Verstärker wirkt auf jeweils zwei Differenzeingänge mit Verstärkungen von 1, 10 bzw. 200.

Als analoge Referenzspannung kann eine interne Referenzspannung von 2,56 V oder die analoge Betriebsspannung AVCC herangezogen werden. Werden die internen Referenzen verwendet, dann ist der Anschluss AREF mit einem Kondensator abzublocken. Die Beschaltung des Anschlusses AREF mit einer externen Referenzspannung wäre ebenfalls möglich, wird aber von stAVeR-24M32 wegen der beschränkten Anzahl zur Verfügung stehender Anschlüsse nicht unterstützt.

Der Eingangsspannungsbereich liegt zwischen 0 V (GND) und der Referenzspannung (- 1 LSB)

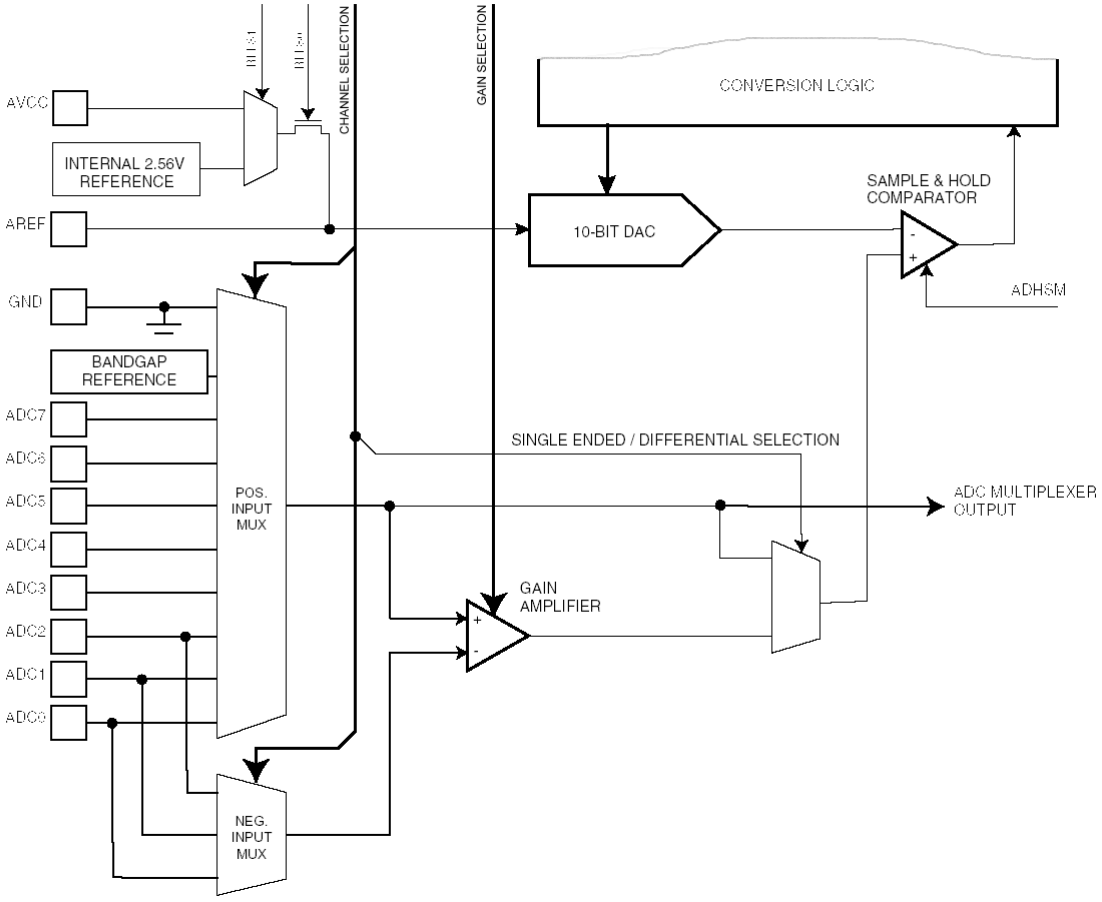


**Abbildung 6 Blockschaltbild AD-Umsetzer**

Für die Steuerung der AD-Umsetzung sind die Register ADMUX, ADCSRA und SFIOR zuständig. Das Ergebnis der AD-Umsetzung steht in den Registern ADCH und ADCL.

| ***ADMUX*** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | REFS1 | REFS0 | ADLAR | MUX4 | MUX3 | MUX2 | MUX1 | MUX0 |
| Resetwert | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Die Bits REFS1 und REFS0 legen die Referenzspannung fest. Nach einem Reset wird eine externe Referenzspannung am Anschluss AVREF erwartet. Tabelle 1 zeigt die Auswahlmöglichkeiten.

| *REFS1* | *REFS0* | *Referenzspannung* |
|---|---|---|
| 0 | 0 | interne Referenzen abgeschaltet |
| 0 | 1 | AVCC mit externem Kondensator an Pin AREF |
| 1 | 0 | reserviert |
| 1 | 1 | interne 2.56V Referenzspannung mit externem Kondensator an Pin AREF |

**Tabelle 1 Auswahl der Referenzspannung**

Das Bit ADLAR legt fest, ob das Ergebnis der AD-Umsetzung linksbündig (ADLAR=1, xxxxxxxxxx000000) oder rechtsbündig (ADLAR=0, 000000xxxxxxxxxx) im 16-Bit Ergebnis abgelegt wird.

Die Bits MUX4:0 programmieren den Analogmultiplexer. Nach Reset ist Eingang ADC0 aktiv. Tabelle 2 zeigt die Auswahlmöglichkeiten bei den massebezogenen Eingängen. Tabelle 3 zeigt die Auswahlmöglichkeiten und Verstärkungsfaktoren bei Differenzeingängen.

| *MUX4:0* | *Eingang gegen GND* |
|---|---|
| 00000 | ADC0 |
| 00001 | ADC1 |
| 00010 | ADC2 |
| 00011 | ADC3 |
| 00100 | ADC4 |
| 00101 | ADC5 |
| 00110 | ADC6 |
| 00111 | ADC7 |
| 11110 | Bandgap-Referenz 1,22 V |
| 11111 | GND 0 V |

**Tabelle 2 Auswahl eines massebezogenen Eingangs**

| *MUX4:0* | *+Eingang* | *-Eingang* | *Verstärkung* |
|---|---|---|---|
| 01000 | ADC0 | ADC0 | 10 |
| 01001 | ADC1 | ADC0 | 10 |
| 01010 | ADC0 | ADC0 | 200 |
| 01011 | ADC1 | ADC0 | 200 |
| 01100 | ADC2 | ADC2 | 10 |
| 01101 | ADC3 | ADC2 | 10 |
| 01110 | ADC2 | ADC2 | 200 |
| 01111 | ADC3 | ADC2 | 200 |
| 10000 | ADC0 | ADC1 | 1 |
| 10001 | ADC1 | ADC1 | 1 |
| 10010 | ADC2 | ADC1 | 1 |
| 10011 | ADC3 | ADC1 | 1 |
| 10100 | ADC4 | ADC1 | 1 |
| 10101 | ADC5 | ADC1 | 1 |
| 10110 | ADC6 | ADC1 | 1 |
| 10111 | ADC7 | ADC1 | 1 |
| 11000 | ADC0 | ADC2 | 1 |
| 11001 | ADC1 | ADC2 | 1 |
| 11010 | ADC2 | ADC2 | 1 |
| 11011 | ADC3 | ADC2 | 1 |
| 11100 | ADC4 | ADC2 | 1 |
| 11101 | ADC5 | ADC2 | 1 |

**Tabelle 3 Auswahl von Differenzeingängen**

| ***ADCSRA*** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | ADEN | ADCS | ADATE | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 |
| Resetwert | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Das Bit ADEN schaltet den AD-Umsetzer ein. Durch das Setzen von Bit ADCS wird eine AD-Umsetzung gestartet. Das Bit bleibt während der Umsetzung gesetzt und wird nach Ende Umsetzung durch die Hardware gelöscht. Das Bit ADATE gibt die im Register SFIOR einzustellende Autotriggerfunktion frei. ADIF ist das AD-Interruptflag und ADIE das AD-Interrupt-Enable. Die Taktfrequenz der sukzessiven Approximation wird aus der Oszillatorfrequenz abgeleitet und durch einen Prescaler bestimmt, der über die Bits ADPS2:0 eingestellt wird (Tabelle 4). Um die maximale Auflösung zu erreichen ist eine Frequenz zwischen 50 kHz und 200 kHz optimal.

| *ADSP2* | *ADSP1* | *ADSP0* | *Prescaler* |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 2 |
| 0 | 1 | 0 | 4 |
| 0 | 1 | 1 | 8 |
| 1 | 0 | 0 | 16 |
| 1 | 0 | 1 | 32 |
| 1 | 1 | 0 | 64 |
| 1 | 1 | 1 | 128 |

**Tabelle 4 Auswahl des Prescalers für die AD-Umsetzung**

| ***SFIOR*** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | ADTS2 | ADTS1 | ADTS0 | ADHSM | ADCME | PUD | PSR2 | PSR10 |
| Resetwert | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Die Bits ADTS2:0 legen die Triggerquelle für die AD-Umsetzung fest, wenn das Bit ADATE im Register ADCSR gesetzt ist. Anderenfalls bleiben sie ohne Einfluss (Tabelle 5).

| ADTS2 | ADTS1 | ADTS0 | Trigger |
|---|---|---|---|
| 0 | 0 | 0 | Free running |
| 0 | 0 | 1 | Analogkomparator |
| 0 | 1 | 0 | Externer Interrupt INT0 |
| 0 | 1 | 1 | Timer/Counter0 Compare Match |
| 1 | 0 | 0 | Timer/Counter0 Overflow |
| 1 | 0 | 1 | Timer/Counter0 Compare Match B |
| 1 | 1 | 0 | Timer/Counter1 Overflow |
| 1 | 1 | 1 | Timer/Counter1 Capture Event |

**Tabelle 5 Auswahl der Triggerquelle der Autotriggerfunktion**

Das Setzen von Bit ADHSM versetzt den AD-Umsetzer in einen High-Speed-Mode, wodurch die Umsetzung auf Kosten der Stromaufnahme beschleunigt wird.

Mit diesen Kenntnissen der internen Register kann der AD-Umsetzer programmiert werden. Für die folgenden Tests habe ich BASCOM-AVR verwendet. In diesem Sammelwerk waren schon häufiger Beiträge die BASCOM-AVR benutzt haben. Die aktuelle Demo-Version von BASCOM-AVR ist neben den Programmbeispielen auf der CD zu finden, kann aber auch vom Entwickler direkt gratis heruntergeladen werden [www.mcselec.com].

# 4. Programmbeispiele

Anhand einiger Programmbeispiele sollen unterschiedliche Betriebsarten des ATmega32 AD-Umsetzers untersucht werden.

In den folgenden Programmbeispielen wird das StAVeR-24M32 Mikrocontrollermodul immer im StAVeR-24M32 Entwicklungsboard eingesetzt. Die Initialisierung des ATmega32 für die dort vorhandene Peripherie erfolgt in einer separaten Initialisierungsdatei, die im nächsten Abschnitt beschrieben wird.

Arbeitet man in einer anderen Hardwareumgebung, dann ist die Initialisierung darauf abzustimmen.

## 4.1. Initialisierung für StAVeR-24M32 Entwicklungsboard

Auf dem StAVeR-24M32 Entwicklungsboard sind ein LCD mit zwei Zeilen zu je sechzehn Zeichen, vier Taster BTN1 bis BTN4 sowie ein Signalgeber (Buzzer) vorhanden. Auf dem StAVeR-24M32 Mikrocontrollermodul befinden sich zusätzlich noch drei LEDs.

Diese Peripherie wird in der Datei init_staver24.bas initialisiert (Listing 1).

```
Init_staver24:
  Config Lcdmode = Port                 ' Setup the LCD correct
  Config Lcd = 16 * 2
  Config Lcdbus = 4
  Config Lcdpin = Pin , Db4 = Porta.4 , Db5 = Porta.5 , Db6 = Porta.6 , Db7 = Porta.7 , E = Portd.6 ,
Rs = Portd.7


  Config Pina.0 = Input                 ' Config these 4 I/O's as inputs (BTN1, BTN2, BTN3 & BTN 4)
  Config Pina.1 = Input
  Config Pina.2 = Input
  Config Pina.3 = Input

  Set Porta.0                           ' Turn on internal PullUp on these 4 pins
  Set Porta.1
  Set Porta.2
  Set Porta.3

  Config Pinb.1 = Output                ' LCD Backlight is connected on this I/O

  Config Pinc.2 = Output                ' GREEN LED
  Config Pinc.3 = Output                ' YELLOW LED
  Config Pinc.4 = Output                ' RED LED

  Config Pind.3 = Output                ' BUZZER

  Set Portd.3                           ' Turn Buzzer OFF
  Set Portb.1                           ' Turn Backlight ON
  Reset Portc.2                         ' Turn on GREEN LED
Return
```

**Listing 1 Initialisierung StAVeR-24M32 (init_staver24.bas)**

Das LCD wird im 4-Bit Mode betrieben. Die Zuordnung der einzelnen Pins wird mit dem Statement Config Lcdpin = ... vorgenommen. Die lange Befehlszeile ist hier für den Druck umgebrochen, muss aber im Quelltext in eine Zeile geschrieben werden!

Es folgt das Definieren der vier Tasteneingänge. Durch das Setzen der betreffenden Portleitungen werden die internen PullUp-Widerstände eingeschaltet.

Schließlich werden noch die Ausgänge definiert, die die LEDs und den Buzzer treiben. Mit einer leuchtenden, grünen LED auf dem StAVeR-24M32 Mikrocontrollermodul wird die Initialisierung verlassen.

Eine Initialisierung des Hardware-UART wird hier nicht vorgenommen.

## 4.2. Programmbeispiele zu AD-Umsetzung

Der AD-Umsetzer des ATmega32 lässt verschiedene Betriebsarten und unterschiedliche Referenzspannungen zu. Außerdem können durch den Analogmultiplexer die verschiedenen Eingangspins an den AD-Umsetzer geführt werden.

Für den Test der unterschiedlichen Betriebsarten in den folgenden Abschnitten habe ich als Eingangsspannung immer die interne Bandgap-Referenz verwendet. Auf diese Weise kennt man das zu erwartende Ergebnis und kann sich auf Konfiguration und Initialisierung konzentrieren.

### 4.2.1. Softwaregetriggerte AD-Umsetzung

BASCOM-AVR unterstützt die internen AD-Umsetzer einiger Bausteine durch eine sehr einfache Konfiguration mit Hilfe der Instruktion `Config Adc =...` . Der Prescaler für die AD-Umsetzung kann automatisch festgelegt werden. Nach dem Start der AD-Umsetzung erfolgt die Abfrage mit der Instruktion getadc().

Die Instruktion `Config Adc = ...` ermöglicht die Auswahl der Betriebsart des AD-Umsetzers (Single Shot oder Free Running), des Prescalers sowie der Referenzspannung.

Bei der Festlegung der Referenzspannung werden alle in Tabelle 1 angegebenen Möglichkeiten berücksichtigt (OFF, AVCC, INTERNAL). Beim stAVeR-24M42 muss jedoch entweder mit AVCC oder der internen Referenzspannung gearbeitet werden, da der Anschluss für eine externe Referenzspannung nicht herausgeführt ist.

Der Prescaler kann auf einen Wert gemäss Tabelle 4 gesetzt werden. Wählt man hingegen den Parameter AUTO, dann sucht der Compiler eine geeigneten Wert aus. Wir werden das noch überprüfen.

Das in Listing 2 gezeigte Programmbeispiel adc1s.bas zeigt eine softwaregetriggerte AD-Umsetzung (SINGLE) bei automatischer Festlegung des Prescalers (AUTO) und Verwendung der analogen Betriebsspannung AVCC als Referenzspannung.

```
'-----------------------------------------------------------
' AD Conversion
' Single-ended  AD conversion using internal reference voltage
'-----------------------------------------------------------
$regfile = "m32def.dat"
$crystal = 14745600
$baud = 19200

$include "init_staver24.bas"

Const Bandgap = &B11110         ' MUX address for bandgap

Dim Result As Word
Dim Voltage As Single
Dim Reference As Single
Dim S As String * 10

Config Adc = Single , Prescaler = Auto , Reference = Avcc

Reference = 5.0                          ' Reference voltage is AVCC
```

```
Start Adc

Do
  Result = Getadc(bandgap)
  Print "ADC  = " ; Hex(result) ; Spc(3);

  Voltage = Result * Reference
  Voltage = Voltage / 1024

  S = Fusing(voltage , " #.###")
  Print "Vref = " ; S ; " V"

  Waitms 500
Loop

End
```

**Listing 2 AD-Umsetzung (adc1s.bas)**

Um die Konfiguration auf Registerebene zu überprüfen bietet sich eine Simulation mit dem internen Simulator an. Im Single-Step können die Registerinhalte bei Abarbeitung der Instruktionen `Config Adc = ...` und `Start Adc` überprüft werden. Das im Atmel Datenblatt mit ADCSRA bezeichnete ADC Controlregister muss wegen der im Registerfile M32DEF.DAT vorgegebenen Definition als ADCSR eingegeben werden.

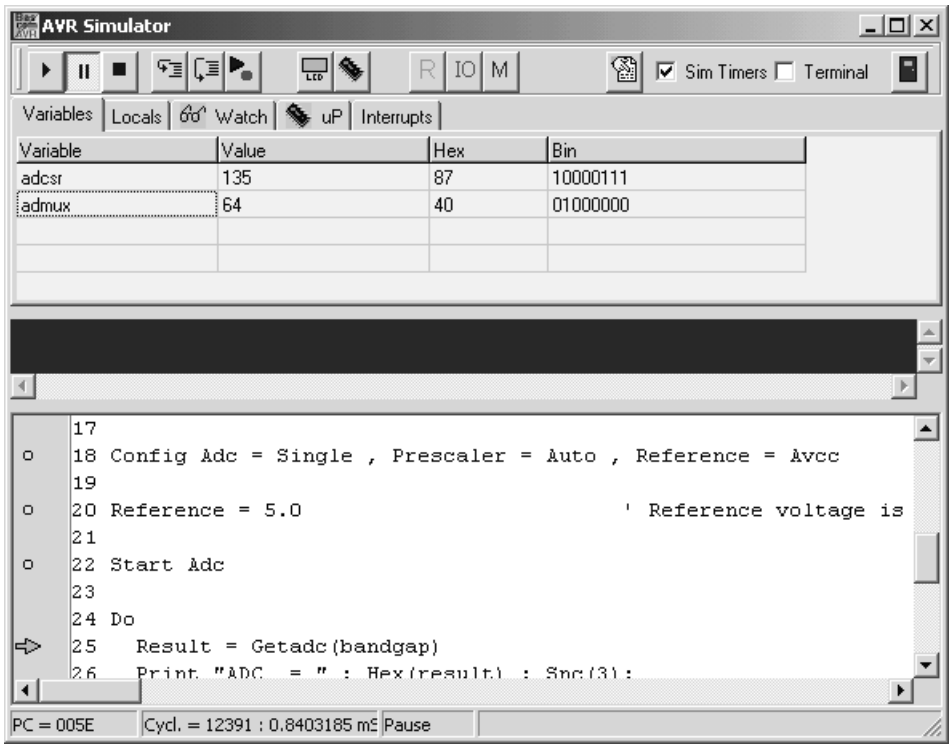


**Abbildung 7 Simulation der AD-Umsetzung**

Aus Abbildung 7 kann entnommen werden, dass ein Prescaler von 128 eingestellt wurde. Das bedeutet bei einer Oszillatorfrequenz von 14,7456 MHz eine Taktfrequenz der AD-Umsetzung von 115,2 kHz. Außerdem setzt die Instruktion `Start Adc` das Bit ADEN und schaltet somit den AD-Umsetzer ein.

Die Abfrage des AD-Umsetzers erfolgt mit der Funktion `Getadc(bandgap)`. Bandgap ist als Konstante definiert und steht für die MUX Adresse der Bandgap-Referenz (siehe Tabelle 2).

Was der Funktionsaufruf im einzelnen bewirkt, erkennt man am besten mit einem Disassembler. Aus dem Funktionsaufruf `result = Getadc(bandgap)` erzeugt der Compiler den folgenden Code:

```
' Result = Getadc(bandgap)
          ldi             r24, 0x5E     ; 004E E58E
          ori             r24, 0x40     ; 004F 6480
          out             $07, r24      ; 0050 B987
          call            sub005E       ; 004D 940E 005E
          ldi             XL, 0x60      ; 004F E6A0
          ldi             XH, 0x00      ; 0050 E0B0
          st              X+, r24       ; 0051 938D
          st              X, r25
          ...
sub005E:  sbi             $06, 6        ; 005E 9A36
avr005F:  sbic            $06, 6        ; 005F 9936
          rjmp            avr005F       ; 0060 CFFE
          sbi             $06, 6        ; 0061 9A36
avr0062:  sbic            $06, 6        ; 0062 9936
          rjmp            avr0062       ; 0063 CFFE
          in              r24, $04      ; 0064 B184
          in              r25, $05      ; 0065 B195
          ret                           ; 0066 9508
```

Aus dem Assemblerlisting kann man erkennen, dass vor der eigentlichen AD-Umsetzung der Wert &H5E in Register ADMUX geschrieben wird. Hierdurch wird der Eingangsmultiplexer für die Bandgap-Referenz durchgeschaltet und AVCC als Referenzspannung ausgewählt. Der folgende Subroutinencall startet zwei AD-Umsetzungen nacheinander.

Eine AD-Umsetzung wird durch Setzen des Bits ADS (`sbi $06, 6`) gestartet. Am Ende der Umsetzung setzte die Hardware das Bit zurück. Das Programm wartet auf diesen Zustand in einer Schleife (`sbic $06,6`).

Am Ende der ersten AD-Umsetzung steht noch kein Resultat in den Registern ADCH und ADCL zur Verfügung. Erst bei der folgenden Umsetzung ist der Resultat verfügbar. Das ist der Grund für die zweiten AD-Umsetzung, die dann das Resultat der ersten bereitstellt. Abbildung 8 zeigt das Timing der AD-Umsetzung.

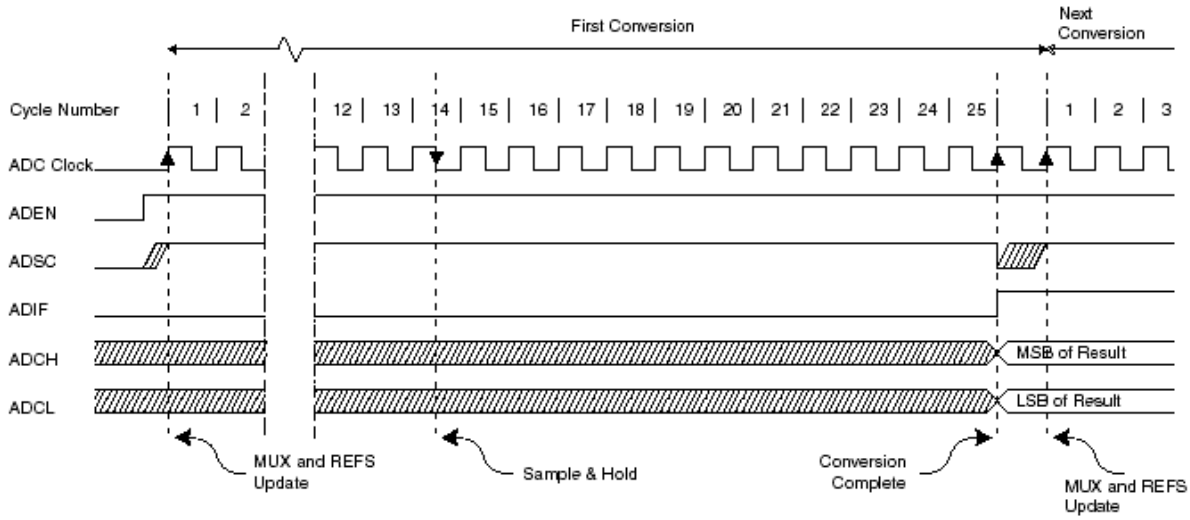


**Abbildung 8 Timing bei der AD-Umsetzung (Single Shot)**

Zum Schluss müssen noch die Register ADCH und ADCL gelesen und in der zugehörigen Variablen abgespeichert werden.

Die in der beschriebenen Weise ermittelten Werte der Bandgap-Referenz werden in einer Schleife des Programms adc1s.bas über die serielle Schnittstelle ausgegeben. Eine eingebaute Wartezeit von 500 ms sorgt dafür, dass ca. zwei Messwerte pro Sekunde erhoben werden.

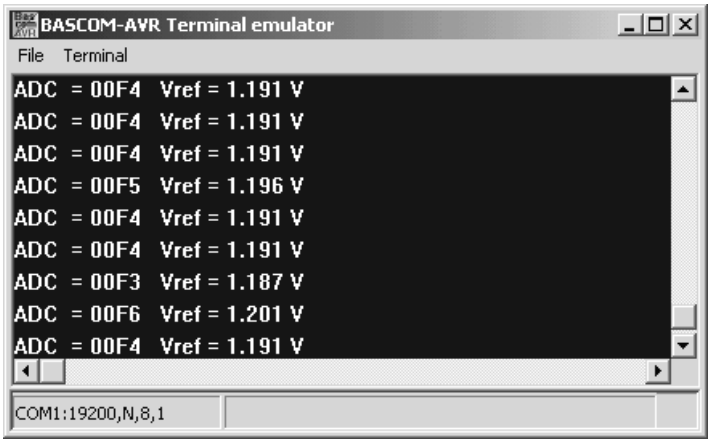


**Abbildung 9 Darstellung der Resultate am Terminal (adc1s.bas)**

### 4.2.2. Free-Running AD-Umsetzung

Beim Ausprobieren des Free-Running-Mode durch `Config Adc = FREE,...` war ich über das identische Assembler-File erst mal erstaunt.

Die Konfigurationsmöglichkeiten der internen Hardware der verschiedenen AVR Mikrocontroller sind aber sehr vielfältig und können nicht in jedem Fall durch die Instruktion Config ... abgedeckt werden. Bringt also die betreffende Config Direktive nicht das gewünschte Ergebnis, dann bleibt immer noch die Möglichkeit, die Register direkt zu initialisieren. Das Datenblatt muss man ohnehin vorher sehr sorgfältig studiert haben.

Also wollen wir den AD-Umsetzer für die Messung der Bandgap-Referenz in den Free-Running-Mode versetzen. Die Register müssen folgendermaßen initialisiert werden:

| | *B7* | *B6* | *B5* | *B4* | *B3* | *B2* | *B1* | *B0* | |
|---|---|---|---|---|---|---|---|---|---|
| ADMUX | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | &H5E |
| ADCSRA | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | &HE7 |
| SFIOR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | &H00 |

Für das Register ADMUX ergibt sich nichts neues. Beim Register ADCSRA wird nun das Bit ADATE gesetzt, was eine interruptgesteuerte Autotriggerfunktion festlegt. Mit den Bits ADTSx (SFIOR7:5) wird die gewünschte Interruptquelle ausgewählt. Für den Free-Running Mode gelten die Defaultwerte nach Reset (SFIOR7:5 = 000), weshalb man sich hierum eigentlich nicht mehr kümmern muss.

Für den Free-Running Mode ist die Interruptquelle der Interrupt ADC Conversion Complete, der durch das Interruptflag ADIF signalisiert wird. Abbildung 10 zeigt das Timing in dieser Betriebsart.

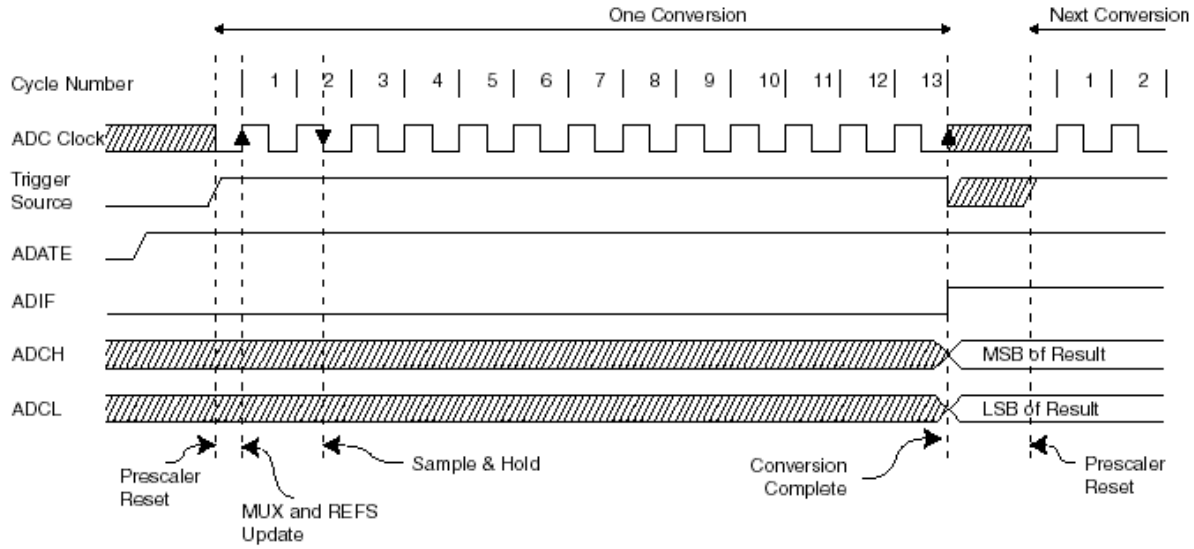


**Abbildung 10 Timing bei der AD-Umsetzung (Free Running)**

Ein erstes Ergebnis liegt wieder mit dem Start der zweiten AD-Umsetzung in den Registern ADCH und ADCL bereit. Die zweite und jede weitere AD-Umsetzung wird durch die Interruptanforderung (ADIF) am Ende der vorangegangenen AD-Umsetzung gestartet. Nur die erste AD-Umsetzung ist wie gehabt durch Setzen des Bits ADS zu starten. Für den Free-Running Mode ist es nicht erforderlich die betreffenden Interrupt Enable Bits zu setzen. Listing 3 zeigt das entsprechend abgeänderte Programmbeispiel.

```
'-----------------------------------------------------------
' AD Conversion
' Single-ended  AD conversion using internal reference voltage
' Free Runnung Mode
'-----------------------------------------------------------
$regfile = "m32def.dat"
$crystal = 14745600
$baud = 19200

$include "init_staver24.bas"

Const Bandgap = &B11110                  ' MUX address for bandgap reference

Dim Result As Word
Dim Temp As Word
Dim Voltage As Single
Dim Reference As Single
Dim S As String * 10

Declare Function Readadc() As Word      ' read ADC result from ADCH & ADCL

' Configuration of ADC for free running mode
Admux = &H5E                             ' AVCC is reference, bandgap is measured
Adcsr = &HA7                             ' prescaler 128, ADC enabled, auto trigger
'Sfior = 0                                ' free running mode (default after reset)

Adcsr = Adcsr Or &H40                    ' start first AD conversion

Reference = 5.0                          ' Reference voltage is AVCC

Do
  Result = Readadc()
  Print "ADC  = " ; Hex(result) ; Spc(3);

  Voltage = Result * Reference
```

```
  Voltage = Voltage / 1024

  S = Fusing(voltage , " #.###")
  Print "Vref = " ; S ; " V"

Loop

End

Function Readadc()                      ' read ADC result from ADCH & ADCL
  Local W1 As Word
  Local W2 As Word

  W1 = Adcl                             ' read Lo byte frist
  W2 = Adch                             ' read Hi byte afterwards
  Shift W2 , Left , 8
  Readadc = W1 + W2
End Function
```

**Listing 3 Free-Running AD-Umsetzung (adc2s.bas)**

Da in dieser Betriebsart die AD-Umsetzung fortlaufend erfolgt, muss das Anwenderprogramm nur noch die Resultate der AD-Umsetzung aus den Ergebnisregistern lesen. Hierzu kann nicht mehr die Funktion `getadc()` verwendet werden, da diese ja auch die AD-Umsetzung startet. Die Funktion `readadc()` liest nur die Register ADCH und ADCL und stellt das Resultat der AD-Umsetzung als 16-Bit Wert zur Verfügung.

Die Initialisierung des AD-Umsetzers erfolgt durch direktes Beschreiben der Register ADMUX und ADCSRA. Der Start der ersten AD-Umsetzung wird hier in einer separaten Anweisung vorgenommen (`Adcsr = Adcsr Or &H40`).

In einer Endlosschleife wird schließlich der AD-Umsetzer abgefragt. Bedingt durch die Formatierungs- und Ausgabeoperationen wird die Funktion `readadc()` ca. alle 10 ms aufgerufen.

### 4.2.3. Timmergetriggerte AD-Umsetzung

In vielen Anwendungen der Messwerterfassung ist eine zeitlich äquidistante Abtastung gefordert. Mit der Autotriggerfunktion des hier betrachteten AD-Umsetzers sind dafür alle Möglichkeiten gegeben.

Im folgenden Programmbeispiel soll der AD-Umsetzer Messwerte im Sekundentakt erfassen. Hierzu wird der AD-Umsetzer wiederum im Autotrigger-Mode betrieben. Gemäss Tabelle 5 kann der Timer1 Overflow als Triggerereignis dienen.

Um einen Sekundentakt zu erzeugen, muss der 16-Bit Timer1 eingesetzt werden. Der Zählbereich von Timer0 umfast nur 8 Bit und ist damit nicht ausreichend.

Abbildung 11 zeigt die Berechnung des Reloadwertes, die sicher auch von Hand durchgeführt werden kann. Das kleine Tool von Jack Tidwell führt aber schnell und komfortabel zum Ziel. Auf der CD ist das Programm zu finden.

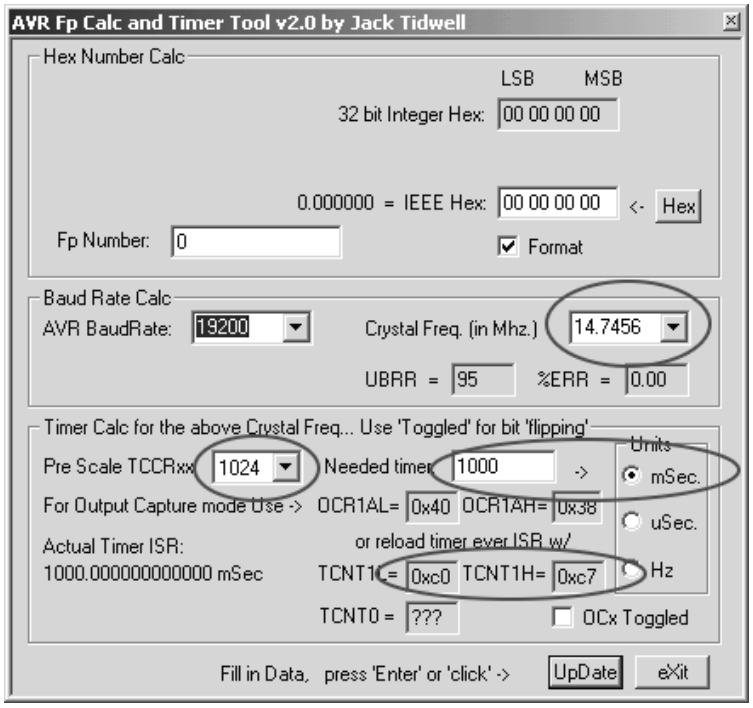


**Abbildung 11 Berechnung des Timer1 Reloads**

Der Sekundentakt soll also den AD-Umsetzer triggern. Das Auslesen der Resultate ist nach Ende der Umsetzung möglich. Hier habe ich den ADC Interrupt verwendet um ein Flag zu setzen, welches in der Hauptschleife ausgewertet wird. Liegt eine neues Resultat einer AD-Umsetzung vor, dann wird es gelesen und über die serielle Schnittstelle auch ausgegeben. Anderenfalls werden im Takt vom 100 ms (`waitms 100`) Punkte ausgegeben, die die Aktivität zwischen den Umsetzungen kennzeichnen sollen.

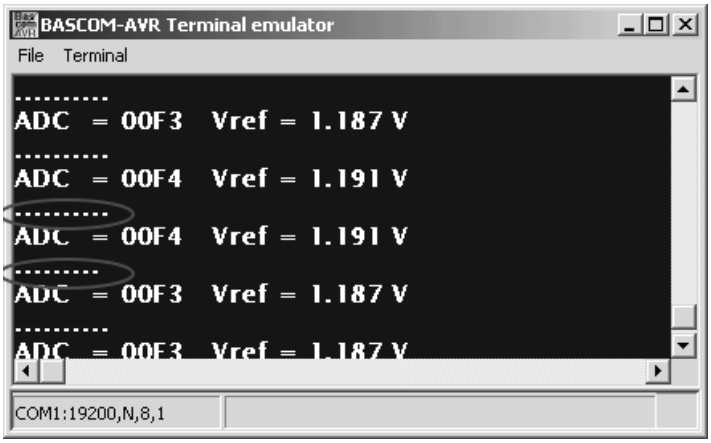


**Abbildung 12 Darstellung der Resultate am Terminal (adc3s.bas)**

Das Programm adc3s.bas (Listing 4) ist mit dem Programm adc2s.bas vergleichbar, nur dass hier mit zwei Interrupt gearbeitet wird.

Die Instruktionen `On Adcc Adcinterrupt` und `On Ovf1 Timerinterrupt` tragen die Interrupt-Serviceroutinen (ISR) Adcinterrupt und Timerinterrupt in die Interruptvektortabelle ein. Am Ende des Listings sind beide ISR zu finden.

Die ISR Timerinterrupt lädt den ermittelten Reloadwert in Timer1, um den Sekundentakt zu sichern. Die ISR Adcinterrupt setzt nur das Flag Adc_ready und lässt für 10 ms die gelbe LED auf dem stAVeR-24M32 aufleuchten.

Die Initialisierung der Register des AD-Umsetzers unterscheidet sich nur bezüglich der ausgewählten Autotrigger-Interruptquelle. Hier wird der Timer1 Overflow verwendet, weshalb die Bits ADTS2:0 im Register SFIOR mit 110 zu laden sind (`Sfior = &B11000000`).

Die Konfiguration von Timer1 erfolgt ganz konventionell, nur dass die Register TCNT1H und TCNT1L mit dem Reloadwert vorgeladen werden.

Nach Freigabe der Interrupts kann die erste AD-Umsetzung gestartet werden und das Programm tritt in die Hauptschleife ein.

In dieser Hauptschleife wird das in der ISR Adcinterrupt gesetzte Flag Adc_ready abgefragt, um entweder eine neues Resultat einer AD-Umsetzung auszulesen, formatiert über die serielle Schnittstelle auszugeben und das Flag zurückzusetzen oder einen "." auszugeben und anschließend 10 ms zu warten. Wie schon Abbildung 12 gezeigt hat, werden in einigen Fällen zehn und in einigen Fällen neun Punkte ausgegeben.

Die Unterschiede haben einige Ursachen. Die Routine `waitms` ist nur annähernd genau und wird durch Interrupts unterbrochen und die Hauptschleife hat eigene Laufzeiten und läuft völlig asynchron zu den Aktivitäten des AD-Umsetzers.

```
'-----------------------------------------------------------
' AD Conversion
' Single-ended  AD conversion using internal reference voltage
' Timer1 triggered
'-----------------------------------------------------------
$regfile = "m32def.dat"
$crystal = 14745600
$baud = 19200

$include "init_staver24.bas"

Yellowled Alias Portc.3
Set Yellowled                   ' switch yellow led off

Const Bandgap = &B11110         ' MUX address for bandgap reference
Const 1_sec_reload = &HC7C0     ' Timer1 relaod value

Dim Result As Word
Dim Temp As Word
Dim Voltage As Single
Dim Reference As Single
Dim S As String * 10
Dim Adc_ready As Bit


Declare Function Readadc() As Word      ' read ADC result from ADCH & ADCL

On Adcc Adcinterrupt            ' interrupt vector for adc ready interrupt
On Ovf1 Timerinterrupt          ' interrupt vector for timer1 overflow interrupt


' Configuration of ADC for autotrigger by timer1 overflow
Admux = &H5E                    ' AVCC is reference, bandgap is measured
Adcsr = &HA7                    ' prescaler 128, ADC enabled, auto trigger
Sfior = &B11000000              ' select timer1 overflow as aut0trigger

Reference = 5.0                 ' Reference voltage is AVCC


Config Timer1 = Timer , Prescale = 1024 ' Timer0 Configuration
Set Sfior.psr10                         ' reset prescaler
```

```
Timer1 = 1_sec_reload

Enable Ovf1                           ' enable timer0 overflow interrupt
Set Adcsr.adie                        ' enable adc interrupt
Enable Interrupts                     ' enable global interrupt

Adcsr = Adcsr Or &H40                 ' start first AD conversion

Do
  If Adc_ready = 1 Then               ' is there a new adc result?
    Result = Readadc()
    Print
    Print "ADC  = " ; Hex(result) ; Spc(3);

    Voltage = Result * Reference
    Voltage = Voltage / 1024

    S = Fusing(voltage , " #.###")
    Print "Vref = " ; S ; " V"

    Adc_ready = 0
  Else
    Print ".";
    Waitms 100
  End If

Loop

End

Function Readadc()                    ' read ADC result from ADCH & ADCL
  Local W1 As Word
  Local W2 As Word

  W1 = Adcl                           ' read Lo byte frist
  W2 = Adch                           ' read Hi byte afterwards
  Shift W2 , Left , 8
  Readadc = W1 + W2
End Function

Timerinterrupt:
  Timer1 = 1_sec_reload
Return

Adcinterrupt:
  Adc_ready = 1                       ' set adc ready flag for main loop
  Reset Yellowled                     ' blink yellow led
  Waitms 10
  Set Yellowled
Return
```

**Listing 4 Timergetriggerte AD-Umsetzung (adc3s.bas)**

# 5. Kennwerte des AD-Umsetzers

## 5.1. Analoge Eingangsschaltung

Wichtig für eingangsseitige Beschaltung des AD-Umsetzers ist die analoge Eingangsschaltung. Abbildung 13 zeigt ein Ersatzschaltbild für die massebezogene Messung.

Die analoge Spannungsquelle (Eingangsspannung) wird in jedem Fall durch die Eingangsleckströme und eine Eingangkapazität des Anschlusses (nicht dargestellt) belastet. Ist der betreffende Multiplexer-Kanal durchgeschaltet, dann muss die Quelle das RC-Glied aus Serienwiderstand und S&H-Kapazität aufladen.

Die Eingangsschaltung ist für Impedanzen von 10 kΩ und weniger optimiert. Liegen die Impedanzen höher, dann kann eine Entkopplung durch einen Operationsverstärker angezeigt sein.

Für die differentielle Messung ist die Eingangsschaltung komplexer. Hier sind Impedanzen von 100 kΩ und weniger günstig.

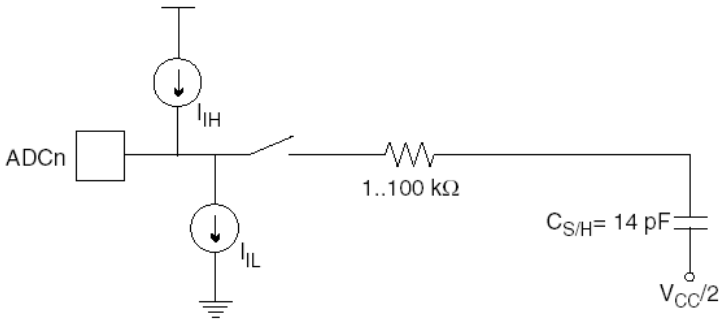


**Abbildung 13 Analoge Eingangsschaltung**

## 5.2. Kennlinie des AD-Umsetzers

Das Resultat der AD-Umsetzung folgt idealerweise für die massebezogene Messung (Single Ended) der Beziehung

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

und für den differentielle Messung der Beziehung

$$V_{ADC} = \frac{(V_{IN+} - V_{IN-}) \cdot GAIN \cdot 512}{V_{REF}}$$

Kaum zu erwarten ist eine fehlerfreie AD-Umsetzung, weshalb den Abweichungen vom Idealverhalten Beachtung geschenkt werden sollte.

Auf diese Weise verlangt man von der ausgewählten Baugruppe nur so viel, wie diese auch zu leisten vermag oder kann die Fehler in bestimmtem Masse eliminieren. Abbildung 14 zeigt die grundsätzlichen Fehler bei der AD-Umsetzung.

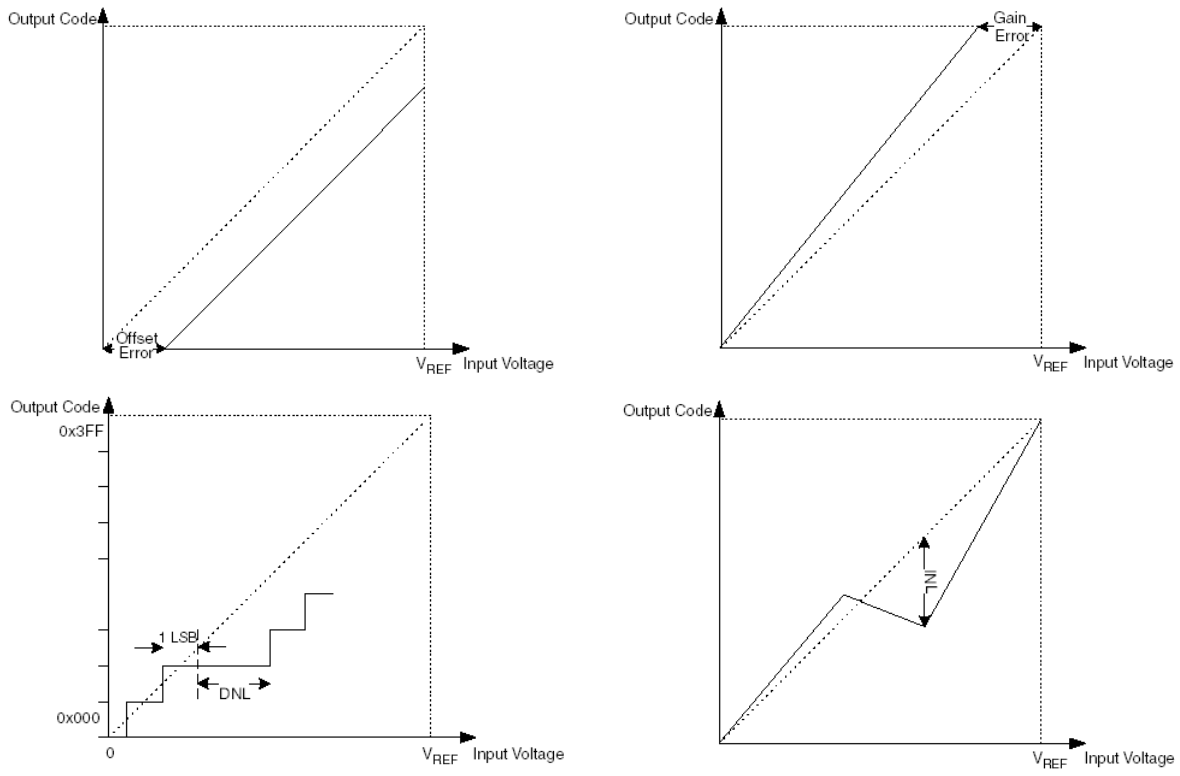


**Abbildung 14 Fehler bei der AD-Umsetzung**

Die Abbildung links oben zeigt einen Offsetfehler. Ein Offsetfehler liegt dann vor, wenn der Übergang des Resultats der AD-Umsetzung von 0 auf 1 nicht bei einer Eingangsspannung von ½ LSB (hier $V_{REF}/2048$) erfolgt. Bei einer Referenzspannung VREF = AVCC = 5 V beträgt das LSB 4,88 mV.

Die Abbildung rechts oben zeigt einen Verstärkungsfehler (Gain Error). Ein Verstärkungsfehler liegt dann vor, wenn der Übergang des Resultats der AD-Umsetzung von $3FE_H$ auf $3FF_H$ nicht bei einer Eingangsspannung 1½ LSB unterhalb der Referenzspannung erfolgt.

Die Abbildung links unten zeigt die differentielle Nichtlinearität (DNL). Die DNL kennzeichnet den maximalen Spannungshub pro Quantisierungsschritt. Ideal ist der Spannungshub für jede Stufe genau 1 LSB.

Die Abbildung rechts unten zeigt die integrale Nichtlinearität (INL). Die INL ist die maximale Abweichung von der Idealkennlinie des AD-Umsetzers.

Neben der statischen Kennlinie des AD-Umsetzers ist auch das Rauschverhalten von Interesse. Hierbei kann aber der AD-Umsetzer nicht isoliert betrachtet werden, sondern die analoge Eingangsschaltung (Leitungslängen etc.) muss Berücksichtigung finden.

Abbildung 15 zeigt ein Histogramm der Umsetzung einer konstanten Eingangsspannung von 2,5 V. Betrachtet wurde ca. 300 nacheinander erhobene Messwerte. Als Analogeingang wurde ADC7 verwendet. Als Referenzspannung diente VACC.

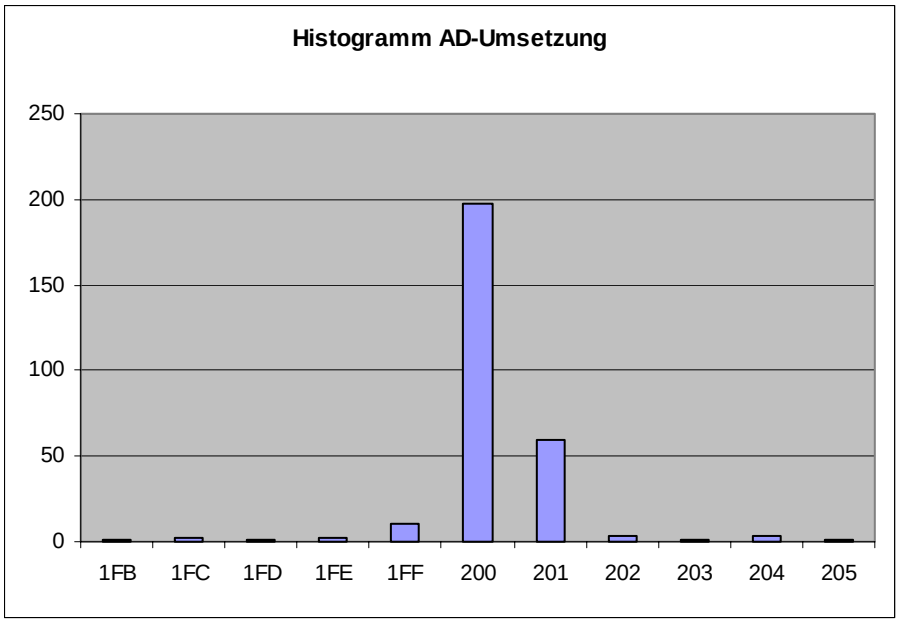


**Abbildung 15 Histogramm der AD-Umsetzung ($V_{IN}$ = 2.5 V)**

Abbildung 15 zeigt die sporadischen Abweichungen in einem Bereich, den man nur ungern tolerieren möchte. Zur Eliminierung solcher Störungen sind die Einflussgrößen detailliert zu untersuchen.

Eine naheliegende Einflussgröße ist mein Testaufbau, der längere Leitungen zur Eingangspannungsquelle (Potentiometer zwischen VDD und GND des stAVeR-24M32) aufwies. Des weiteren wäre ein genauerer Blick auf die Güte der als Referenzspannung dienenden analogen Betriebsspannung AVCC auf dem stAVeR-24M32 Modul erforderlich.

Für einen Test der unterschiedlichen Betriebsarten reicht ein solcher Aufbau – für eine quantitative Analyse der Kennwerte eines solchen AD-Umsetzers eher nicht.

# 6. Links

**StAVeR Microcontroller Module (engl.)**

http://www.lawicel.com/staver/

**Zahlreiche Informationen zu AVR Mikrocontrollern (engl.)**

http://www.avrfreaks.net/

**Website von MCS Electronics - Entwickler von BASCOM (engl.)**

http://www.mcselec.com

**Website des Autors**

http://www.ckuehnel.ch

# 7. Literatur

Ruscak, S.; Singer, L.:
Using Histogram Techniques to Measure A/D Converter Noise
Analog Dialogue, Vol. 29, No. 2, 1995

Application Note AN-351
Trimming Offset and Gain in A/D's and D/A's
www.analog.com