

```

#####
; Einfacher 8 Bit Logic Analyzer zur interaktiven Nutzung per PC zu Platine TN20190328
; Hardware: ATMEGA328 @16MHz, externer Quarz an PB6 & PB7, kein ISP-Interface
; Sämtliche Fuse Bytes auf FFh (unprogrammed)
; Port C: RS232 Interface gemäß AppNote305 - PC0 = RxD, PC1 = TxD, Parameter 19200/8/N/1
; Port B: PB0 = Sample LED, PB1 = TxD LED, Port D: Liest die Eingangssignale (8 Bit)
;
; V1.20: 02/06/2019 - Code-Umfang 246 Byte & 28 Datenbytes, (c) T. Neveling
#####
#include "m328Pdef.inc"          ; Definitionsdatei für ATmega328P µC einbinden
;
; Konstanten Definitionen
.equ    RxD      = 0              ; RS232 Receive Pin = PC0
.equ    TxD      = 1              ; RS232 Transmit Pin = PC1
.equ    sb       = 1              ; RS232 Stop Bits (1, 2, ...)
.equ    b        = 135           ; Delay Konstante nach AN305 für 19200Bd @16MHz Takt
.equ    pretrig   = 25            ; Trigger Vorlauf
;
; Register-Zuordnung
.def    NULL     = r14            ; Immer 0
.def    Trigger   = r15           ; Triggermaske
.def    temp      = r16           ; Universalregister
.def    BitCnt    = r17           ; zur seriellen Datenübertragung per AppNote 305
.def    TxByte    = r18           ; RS232: Zu sendendes Byte
.def    RxByte    = r19           ; RS232: Empfangenes Byte
.def    SampTme   = r20           ; Code für Sampleintervalle 1µs-16ms von PC GUI via RS232
.def    PreScale  = r21           ; Inhalt von TCCR0B für Prescaler 1:1, 1:8, 1:64...
.def    CompWert  = r22           ; Timer0 Compare A Wert für gewünschtes Sample Intervall
.def    Flags     = r23           ; diverse Stati
.equ    FoundTrg  = 0             ; Bit 0: Triggerbedingung erfüllt
;
; r26/27 = xl/xh adressiert das SRAM
; r28/29 = yl/yh für die Triggerposition
; r30/31 = zl/zh zur Abfrage der Initialisierungs Tabelle für Timer0
;
.cseg                             ; Codesegment
;
; Interrupt Vektoren
.org 0
    rjmp start                    ; Reset Handler, Startadresse 0
;
; Initalisierung
start:
    ldi temp, LOW(RAMEND)         ; Stackpointer setzen
    out SPL, temp                ; LOW-Byte der obersten RAM-Adresse
    ldi temp, HIGH(RAMEND)        ; HIGH-Byte der obersten RAM-Adresse
    out SPH, temp                 ; eigentlich default auf RAMEND, siehe Datenblatt S.12
;
; Variablen initialisieren
    clr NULL
;
; Ports initialisieren: "1" = Aushang, "0" = Eingang
    sbi DDRC, TxD                 ; TxD (PC1) auf Ausgang & High initialisieren
    sbi PORTC, TxD
    ldi temp, 0b00000011          ; LED Ausgänge an PB0 & PB1 (schalten gegen Masse)
    out DDRB, temp
    out PORTB, NULL               ; LEDs aus
    out DDRD, NULL               ; Port D: 8 Signaleingänge mit HW Pull-down Widerständen
;
; Timer0 initialisieren
    ldi temp, 1<<WGM01            ; CTC Modus: Clear Counter on Compare Match
    out TCCR0A, temp
    out TCCR0B, NULL             ; Timer0 zunächst stoppen
;
main:
    rcall GetChar                 ; Modi per RS232 abfragen und Analyzer starten
    mov SampTme, RxByte           ; Code für Samplerate lesen (0 .. 13, 14 = Testpattern)
    rcall GetChar                 ; 0 = 1µs, 1 = 2µs etc, 14 = Testmodus
    mov Trigger, RxByte           ; Trigger Maske muss mit gesampeltem Byte übereinstimmen
;
    cpi SampTme, 14               ; Code für binäres Testmuster
    brne SetTimer0
    rcall TestPattern
    rjmp main                    ; Endlos Schleife
;
SetTimer0:
;
; Werte für Timer0 Prescaler (TCCR0B) und CompWert (OCR0A) aus Tabelle SampleTab holen

```

```

    ldi z1, low(SampleTab*2)          ; Zeiger auf Tabellenbeginn
    ldi zh, high(SampleTab*2)
    lsl SampTme                      ; *2, da 2 Byte je Eintrag (CompWert & Prescale)
    add z1, SampTme                  ; Zeiger jetzt auf CompWert des gewünschten Eintrags
    adc zh, NULL                     ; eigentlich unnötig, da Addr(SampleTab) < 256 Byte
    lpm CompWert, z+                 ; CompWert laden (für OCR0A) mit post Inkrement Zeiger
    lpm Prescale, z                  ; Prescale laden (für TCCR0B)
    rcall StartTrace                 ; 2000 Byte einlesen und per RS232 an PC senden
    rjmp main                        ; Endlos Schleife
;
SampleTab: ; =====[ Tabelle zur Timer0 Initialisierung ]=====
;
;   CompWert Prescaler
;   .db 16, 0b00000001          ; 1µs => Prescaler = 1:1    => 16 Takte zu 16MHz
;   .db 32, 0b00000001          ; 2µs => Prescaler = 1:1    => 32 Takte zu 16MHz
;   .db 80, 0b00000001          ; 5µs => Prescaler = 1:1    => 80 Takte zu 16MHz
;   .db 160, 0b00000001         ; 10µs => Prescaler = 1:1   => 160 Takte zu 16MHz
;   .db 40, 0b00000010          ; 20µs => Prescaler = 1:8   => 40 Takte zu 2MHz
;   .db 100, 0b00000010         ; 50µs => Prescaler = 1:8   => 100 Takte zu 2MHz
;   .db 200, 0b00000010         ; 100µs => Prescaler = 1:8  => 200 Takte zu 2MHz
;   .db 50, 0b00000011          ; 200µs => Prescaler = 1:64  => 50 Takte zu 250kHz
;   .db 125, 0b00000011         ; 500µs => Prescaler = 1:64 => 125 Takte zu 250kHz
;   .db 250, 0b00000011         ; 1ms => Prescaler = 1:64   => 250 Takte zu 250kHz
;   .db 125, 0b000000100        ; 2ms => Prescaler = 1:256  => 125 Takte zu 62500Hz
;   .db 250, 0b000000100        ; 4ms => Prescaler = 1:256  => 250 Takte zu 62500Hz
;   .db 125, 0b000000101        ; 8ms => Prescaler = 1:1024 => 125 Takte zu 15625Hz
;   .db 250, 0b000000101        ; 16ms => Prescaler = 1:1024 => 250 Takte zu 15625Hz
;
;=====[ binäres Zählmuster im SRAM erzeugen ]=====
TestPattern:
    clr temp                        ; Zählvariable
    ldi xl, Low (SRAM START)        ; Zeiger auf SRAM Beginn = 100H (für späteres ST X+,temp)
    ldi xh, High (SRAM START)
TestLoop:
    st x+, temp                     ; SRAM mit Binärzähler voll schreiben
    inc temp
    cpi xh, 8                        ; 100h bis 8D0h = 2000 Byte Puffer gefüllt ?
    brlo TestLoop
    cpi xl, 0xD0
    brlo TestLoop
;
    ldi yl, 01                      ; Triggermarke = 257. 100h werden in GUI wieder abgezogen
    ldi yh, 01                      ; 1 wird in SendBuffer abgezogen => Anzeige = 0
    rcall SendBuffer
    ret
;
;=====[ Messung durchführen ]=====
StartTrace:
    ldi xl, Low (SRAM START)        ; Zeiger auf SRAM Beginn = 100H (für späteres ST X+,temp)
    ldi xh, High (SRAM START)
    ldi yl, 0x01                    ; initiale Triggermarke = 901h = 2305 jenseits SRAM Range
    ldi yh, 0x09                    ; GUI gibt 800h= 2048 an, wenn Trigger nicht erkannt wird
    cbr Flags, 1<<FoundTrg         ; Triggerflag löschen
;
    out OCR0A, CompWert              ; Prescaler / Compare Wert für gewünschte Samplettime
    out TCCR0B, Prescale            ; Timer0 starten
    sbi PORTB, 0                    ; Sample LED ein
;
Get Samples:
    sbis TIFR0, OCF0A               ; Compare Matches verarbeiten bis der Puffer voll ist
    rjmp Get_Samples               ; #1 warte bis Compare Match Flag in TIFR0 gesetzt ist
    ; #2
;
    in temp, PIND                   ; #1 Sampling Zeitpunkt erreicht => Eingangspattern lesen
    st x+, temp                     ; #2 und im SRAM speichern mit Post Inkrement SRAM Zeiger
    sbi TIFR0, OCF0A               ; #2 Output Compare Flag wieder löschen
;
    sbrc Flags, FoundTrg            ; #1 wenn Triggerpunkt schon gefunden wurde
    rjmp check_Buffer              ; #2 keine weitere Aktion
;
    cpi xh, 2                       ; #1 Trigger Vorlauf 1..255 "pretrig" abwarten
    brsh check_Trigger             ; #1/2 oberhalb von 255 immer prüfen
    cpi xl, pretrig                ; #1 xh = 1 => Trigger Vorlauf abwarten
    brlo check_Buffer              ; #1/2 nicht prüfen wenn < pretrig
;
check_Trigger:
    cpse temp, Trigger              ; #1 sonst prüfen: Triggermaske = Eingangspattern ?
    rjmp check_Buffer              ; #2 Sprung, wenn nicht
;

```

```

    sbr Flags, 1<<FoundTrg      ; #1 Triggerflag setzen
    mov yl, xl                  ; #1 Triggeradresse im SRAM merken
    mov yh, xh                  ; #1
;                               ; in der GUI dann nochmal 100h = 256 abziehen SRAM_START
check Buffer:
    cpi xh, 8                   ; #1 100h bis 8D0h = 2000 Byte Puffer gefüllt ?
    brlo Get_Samples           ; #1/2 SRAM = 2048 Byte => 48 Byte bleiben für Stack
    cpi xl, 0xD0                ; #1
    brlo Get_Samples           ; #1/2
;
    out TCCR0B, NULL            ; Timer stoppen - SAMPLING BEENDET
    cbi PORTB, 0                ; sample LED aus
;
; Pufferinhalt an PC Senden
SendBuffer:
    sbi PORTB, 1                ; Tx LED ein
    sbiw YH:YL, 1               ; letztes post Inkrement aus St X+ zurücknehmen
    mov TxByte, yl              ; Triggermarke senden
    rcall putchar
    mov TxByte, yh
    rcall putchar
    ldi xl, Low (SRAM_START)     ; Zeiger auf SRAM Beginn (für späteres LD TxByte, X+)
    ldi xh, High (SRAM_START)
SendLoop:
    ld TxByte, X+
    rcall putchar
    cpi xh, 8                    ; 2000 Byte senden
    brlo SendLoop
    cpi xl, 0xD0
    brlo SendLoop
    cbi PORTB, 1                ; Tx LED aus
    ret
;
;*****
; Unterprogramme zur seriellen Kommunikation nach ATMEL AppNote 305
;*****
; "putchar"
;
; * This subroutine transmits the byte stored in the "TxByte" register
; * The number of stop bits used is set with the sb constant
; *
; * Number of words:      14 including return
; * Number of cycles:     Depends on bit rate
; * Low registers used:   None
; * High registers used:  2 (BitCnt, TxByte)
; * Pointers used:        None
;*****

putchar:
    ldi BitCnt, 9+sb            ; 1+8+sb (sb is # of stop bits)
    com TxByte                  ; Invert everything
    sec                         ; Start bit
;
putchar0:
    brcc putchar1              ; If carry set
    cbi PORTC, TxD              ; send a '0'
    rjmp putchar2              ; else
putchar1:
    sbi PORTC, TxD              ; send a '1'
    nop
;
putchar2:
    rcall UART_delay            ; One bit delay
    rcall UART_delay
;
    lsr TxByte                  ; Get next bit
    dec BitCnt                  ; If not all bit sent
    brne putchar0              ; send next
    ret                         ; else return
;
;*****
; "getchar"
;
; * This subroutine receives one byte and returns it in the "RxByte" register
; *
; * Number of words:      14 including return
; * Number of cycles:     Depends on when data arrives
; * Low registers used:   None

```

```
; * High registers used: 2 (BitCnt,RxByte)
; * Pointers used:      None
; *****
;
getchar:
    ldi BitCnt,9                ; 8 data bit + 1 stop bit
;
getchar1:
    sbic PINC,RxD              ; Wait for start bit
    rjmp getchar1
;
    rcall UART_delay           ; 0.5 bit delay
;
getchar2:
    rcall UART_delay           ; 1 bit delay
    rcall UART_delay
;
    clc                        ; clear carry
    sbic PINC,RxD              ; if RX pin high
    sec
;
    dec BitCnt                 ; If bit is stop bit
    breq getchar3              ; return
                                ; else
    ror RxByte                 ; shift bit into Rxbyte
    rjmp getchar2              ; go get next
;
getchar3:
    ret

; *****
; * "UART_delay"
; *
; * This delay subroutine generates the required delay between the bits when
; * transmitting and receiving bytes. The total execution time is set by the
; * constant "b":
; *
; * Number of words:      4 including return
; * Low registers used:   None
; * High registers used:  1 (temp)
; * Pointers used:        None
; *****
; Berechnung von b, siehe Appnote documentation
;
;  $b = (F_{CPU} / \text{Baud} - 23) / 6$ 
; für 16MHz Quarztakt und 19200 Baud:
;  $b = (16000000 / 19200 - 23) / 6 = (833,33 - 23) / 6 = 135,05 \sim 135$ 
;
UART_delay:
    ldi temp,b
UART_delay1:
    dec temp
    brne UART_delay1
    ret
```