```
;***************************************************************************
;* U S B   S T A C K   F O R   T H E   A V R   F A M I L Y
;*
;* File Name            :"USBtoRS232.asm"
;* Title                :AVR309:USB to UART protocol converter
;* Date                 :01.02.2004
;* Version              :2.8
;* Target MCU           :ATmega8
;* AUTHOR               :Ing. Igor Cesko
;*                       Slovakia
;*                       cesko@internet.sk
;*                       http://www.cesko.host.sk
;*
;* DESCRIPTION:
;*  USB protocol implementation into MCU with noUSB interface:
;*  Device:
;*  Universal USB interface (3x8-bit I/O port + RS232 serial line + EEPROM)
;*  + added RS232 FIFO buffer
;*
;* The timing is adapted for 12 MHz crystal
;*
;*
;* to add your own functions - see section: TEMPLATE OF YOUR FUNCTION
;*
;* to customize device to your company you must change VendorUSB ID (VID)
;* to VID assigned to your company (for more information see www.usb.org)
;*
;***************************************************************************
.include "m8def.inc"
;comment for AT90S2313
.equ    UCR                     =UCSRB
.equ    UBRR                    =UBRRL
.equ    EEAR                    =EEARL
.equ    USR                     =UCSRA
.equ    E2END                   =127
.equ    RAMEND128               =96+127

.equ    inputport               =PINB
.equ    outputport              =PORTB
.equ    USBdirection            =DDRB
.equ    DATAplus                =1              ;signal D+ on PB1
.equ    DATAminus               =0              ;signal D- on PB0 - give on this pin pull-up 1.5kOhm
.equ    USBpinmask              =0b11111100     ;mask low 2 bit (D+,D-) on PB
.equ    USBpinmaskDplus         =~(1<<DATAplus) ;mask D+ bit on PB
.equ    USBpinmaskDminus        =~(1<<DATAminus);mask D- bit on PB

.equ    TSOPPort                =PINB
.equ    TSOPpullupPort          =PORTB
.equ    TSOPPin                 =2              ;signal OUT from IR sensor TSOP1738 on PB2

;connecting LED diode LSB
;connecting LED diode LSB (input)
;input/output LED LSB
;connecting LED diode MSB
;connecting LED diode MSB  (input)
;input/output LED MSB
```

```
;LED0 on pin PD3
;LED1 on pin PD5
;LED2 on pin PD6
;LED3 on pin PB3
;LED4 on pin PB4
;LED5 on pin PB5
;LED6 on pin PB6
;LED7 on pin PB7

.equ    SOPbyte                 =0b10000000     ;Start of Packet byte
.equ    DATA0PID                =0b11000011     ;PID for DATA0 field
.equ    DATA1PID                =0b01001011     ;PID for DATA1 field
.equ    OUTPID                  =0b11100001     ;PID for OUT field
.equ    INPID                   =0b01101001     ;PID for IN field
.equ    SOFPID                  =0b10100101     ;PID for SOF field
.equ    SETUPPID                =0b00101101     ;PID for SETUP field
.equ    ACKPID                  =0b11010010     ;PID for ACK field
.equ    NAKPID                  =0b01011010     ;PID for NAK field
.equ    STALLPID                =0b00011110     ;PID for STALL field
.equ    PREPID                  =0b00111100     ;PID for FOR field

.equ    nSOPbyte                =0b00000001     ;Start of Packet byte - reverse order
.equ    nDATA0PID               =0b11000011     ;PID for DATA0 field - reverse order
.equ    nDATA1PID               =0b11010010     ;PID for DATA1 field - reverse order
.equ    nOUTPID                 =0b10000111     ;PID for OUT field - reverse order
.equ    nINPID                  =0b10010110     ;PID for IN field - reverse order
.equ    nSOFPID                 =0b10100101     ;PID for SOF field - reverse order
.equ    nSETUPPID               =0b10110100     ;PID for SETUP field - reverse order
.equ    nACKPID                 =0b01001011     ;PID for ACK field - reverse order
.equ    nNAKPID                 =0b01011010     ;PID for NAK field - reverse order
.equ    nSTALLPID               =0b01111000     ;PID for STALL field - reverse order
.equ    nPREPID                 =0b00111100     ;PID for FOR field - reverse order

.equ    nNRZITokenPID           =~0b10000000    ;PID mask for Token packet (IN,OUT,SOF,SETUP) - reverse order NRZI
.equ    nNRZISOPbyte            =~0b10101011    ;Start of Packet byte - reverse order NRZI
.equ    nNRZIDATA0PID           =~0b11010111    ;PID for DATA0 field - reverse order NRZI
.equ    nNRZIDATA1PID           =~0b11001001    ;PID for DATA1 field - reverse order NRZI
.equ    nNRZIOUTPID             =~0b10101111    ;PID for OUT field - reverse order NRZI
.equ    nNRZIINPID              =~0b10110001    ;PID for IN field - reverse order NRZI
.equ    nNRZISOFPID             =~0b10010011    ;PID for SOF field - reverse order NRZI
.equ    nNRZISETUPPID           =~0b10001101    ;PID for SETUP field - reverse order NRZI
.equ    nNRZIACKPID             =~0b00100111    ;PID for ACK field - reverse order NRZI
.equ    nNRZINAKPID             =~0b00111001    ;PID for NAK field - reverse order NRZI
.equ    nNRZISTALLPID           =~0b00000111    ;PID for STALL field - reverse order NRZI
.equ    nNRZIPREPID             =~0b01111101    ;PID for FOR field - reverse order NRZI
.equ    nNRZIADDR0              =~0b01010101    ;Address = 0 - reverse order NRZI

                                                ;status bytes - State
.equ    BaseState               =0              ;
.equ    SetupState              =1              ;
.equ    InState                 =2              ;
.equ    OutState                =3              ;
.equ    SOFState                =4              ;
.equ    DataState               =5              ;
.equ    AddressChangeState      =6              ;

```

```
                                                ;Flags of action
.equ    DoNone                                  =0
.equ    DoReceiveOutData                        =1
.equ    DoReceiveSetupData                      =2
.equ    DoPrepareOutContinuousBuffer            =3
.equ    DoReadySendAnswer                       =4


.equ    CRC5poly                =0b00101                ;CRC5 polynomial
.equ    CRC5zvysok              =0b01100                ;CRC5 remainder after successful CRC5
.equ    CRC16poly               =0b1000000000000101     ;CRC16 polynomial
.equ    CRC16zvysok             =0b1000000000001101     ;CRC16 remainder after successful CRC16

.equ    MAXUSBBYTES             =14                     ;maximum bytes in USB input message
.equ    NumberOfFirstBits       =10                     ;how many first bits allowed be longer
.equ    NoFirstBitsTimerOffset  =256-12800*12/1024      ;Timeout 12.8ms (12800us) to terminate after firsts bits
.equ    InitBaudRate            =12000000/16/57600-1    ;UART on 57600 (for 12MHz=12000000Hz)

.equ    InputBufferBegin        =RAMEND128-127                          ;compare of receiving shift buffer
.equ    InputShiftBufferBegin   =InputBufferBegin+MAXUSBBYTES           ;compare of receiving buffera

.equ    MyInAddressSRAM         =InputShiftBufferBegin+MAXUSBBYTES
.equ    MyOutAddressSRAM        =MyInAddressSRAM+1

.equ    OutputBufferBegin       =RAMEND128-MAXUSBBYTES-2        ;compare of transmitting buffer
.equ    AckBufferBegin          =OutputBufferBegin-3    ;compare of transmitting buffer Ack
.equ    NakBufferBegin          =AckBufferBegin-3       ;compare of transmitting buffer Nak
.equ    ConfigByte              =NakBufferBegin-1       ;0=unconfigured state
.equ    AnswerArray             =ConfigByte-8           ;8 byte answer array
.equ    StackBegin              =AnswerArray-1          ;low reservoir (stack is big cca 68 byte)

.equ    MAXRS232LENGTH          =RAMEND-RAMEND128-10    ;maximum length RS232 code
.equ    RS232BufferBegin        =RAMEND128+1            ;compare of buffer for RS232 - receiving
.equ    RS232BufferEnd          =RS232BufferBegin+MAXRS232LENGTH
.equ    RS232ReadPosPtr         =RS232BufferBegin+0
.equ    RS232WritePosPtr        =RS232BufferBegin+2
.equ    RS232LengthPosPtr       =RS232BufferBegin+4
.equ    RS232Reserved           =RS232BufferBegin+6
.equ    RS232FIFOBegin          =RS232BufferBegin+8



.def    RS232BufferFull         =R1             ;flag of full RS232 buffer
.def    backupbitcount          =R2             ;backup bitcount register in INT0 disconnected
.def    RAMread                 =R3             ;if reading from SRAM
.def    backupSREGTimer         =R4             ;backup Flag register in Timer interrupt
.def    backupSREG              =R5             ;backup Flag register in INT0 interrupt
.def    ACC                     =R6             ;accumulator
.def    lastBitstufNumber       =R7             ;position in bitstuffing
.def    OutBitStuffNumber       =R8             ;how many bits to send last byte - bitstuffing
.def    BitStuffInOut           =R9             ;if insertion or deleting of bitstuffing
.def    TotalBytesToSend        =R10            ;how many bytes to send
.def    TransmitPart            =R11            ;order number of transmitting part
.def    InputBufferLength       =R12            ;length prepared in input USB buffer
.def    OutputBufferLength      =R13            ;length answers prepared in USB buffer
.def    MyOutAddress            =R14            ;my USB address (Out Packet) for update
```

```
.def    MyInAddress             =R15            ;my USB address (In/SetupPacket)


.def    ActionFlag              =R16            ;what to do in main program loop
.def    temp3                   =R17            ;temporary register
.def    temp2                   =R18            ;temporary register
.def    temp1                   =R19            ;temporary register
.def    temp0                   =R20            ;temporary register
.def    bitcount                =R21            ;counter of bits in byte
.def    ByteCount               =R22            ;counter of maximum number of received bytes
.def    inputbuf                =R23            ;receiver register
.def    shiftbuf                =R24            ;shift receiving register
.def    State                   =R25            ;state byte of status of state machine
.def    RS232BufptrX            =R26            ;XL register - pointer to buffer of received IR codes
.def    RS232BufptrXH           =R27
.def    USBBufptrY              =R28            ;YL register - pointer to USB buffer input/output
.def    ROMBufptrZ              =R30            ;ZL register - pointer to buffer of ROM data


;requirements on descriptors
.equ    GET_STATUS              =0
.equ    CLEAR_FEATURE           =1
.equ    SET_FEATURE             =3
.equ    SET_ADDRESS             =5
.equ    GET_DESCRIPTOR          =6
.equ    SET_DESCRIPTOR          =7
.equ    GET_CONFIGURATION       =8
.equ    SET_CONFIGURATION       =9
.equ    GET_INTERFACE           =10
.equ    SET_INTERFACE           =11
.equ    SYNCH_FRAME             =12

;descriptor types
.equ    DEVICE                  =1
.equ    CONFIGURATION           =2
.equ    STRING                  =3
.equ    INTERFACE               =4
.equ    ENDPOINT                =5

;databits
.equ    DataBits5               =0
.equ    DataBits6               =1
.equ    DataBits7               =2
.equ    DataBits8               =3

;parity
.equ    ParityNone              =0
.equ    ParityOdd               =1
.equ    ParityEven              =2
.equ    ParityMark              =3
.equ    ParitySpace             =4

;stopbits
.equ    StopBit1                =0
.equ    StopBit2                =1

```

```
;user function start number
.equ    USER_FNC_NUMBER         =100


;------------------------------------------------------------------------------------------
;*********************************************************************
;* Interrupt table
;*********************************************************************
.cseg
;------------------------------------------------------------------------------------------
.org 0                                          ;after reset
                rjmp    reset
;------------------------------------------------------------------------------------------
.org INT0addr                                   ;external interrupt INT0
                rjmp    INT0handler
;------------------------------------------------------------------------------------------
.org URXCaddr                                   ;receiving from serial line
                push    temp0
                cbi     UCR,RXCIE                       ;disable interrupt from UART receiving
                sei                                     ;enable interrupts to service USB
                in      temp0,UDR                       ;put to temp0 received data from UART
                in      backupSREGTimer,SREG            ;backup SREG
                push    temp2
                push    temp3
                lds     temp2,RS232LengthPosPtr
                lds     temp3,RS232LengthPosPtr+1               ;determine length of RS232 code buffer
                cpi     temp3,HIGH(RS232BufferEnd-RS232FIFOBegin-1)     ;if the buffer would overflow
                brlo    FIFOBufferNoOverflow                    ;if not overflow then write to FIFO
                brne    NoIncRS232BufferFull                    ;if buffer would overflow, then prevent of overwriting
                                                                ;otherwise (if equall) still compare Lo bytes
                cpi     temp2,LOW(RS232BufferEnd-RS232FIFOBegin-1)      ;if buffer would overflow (Lo byte)
                brcc    NoIncRS232BufferFull                    ;then prevent of overwriting
FIFOBufferNoOverflow:
                push    RS232BufptrX
                push    RS232BufptrXH
                lds     RS232BufptrX,RS232WritePosPtr           ;set position to begin of buffer write RS232 code
                lds     RS232BufptrXH,RS232WritePosPtr+1        ;set position to begin of buffer write RS232 code

                st      X+,temp0                                ;and save it to buffer
                cpi     RS232BufptrXH,HIGH(RS232BufferEnd+1)    ;if not reached maximum of RS232 buffer
                brlo    NoUARTBufferOverflow                    ;then continue
                brne    UARTBufferOverflow                      ;check althen LSB
                cpi     RS232BufptrX,LOW(RS232BufferEnd+1)      ;if not reached maximum of RS232 buffer
                brlo    NoUARTBufferOverflow                    ;then continue
 UARTBufferOverflow:
                ldi     RS232BufptrX,LOW(RS232FIFOBegin)        ;otherwise set position to buffer begin
                ldi     RS232BufptrXH,HIGH(RS232FIFOBegin)      ;otherwise set position to buffer begin
 NoUARTBufferOverflow:
                sts     RS232WritePosPtr,RS232BufptrX           ;save new offset of buffer write RS232 code
                sts     RS232WritePosPtr+1,RS232BufptrXH        ;save new offset of buffer write RS232 code
                ldi     temp0,1                                 ;increment length of RS232 buffer
                add     temp2,temp0
                ldi     temp0,0
                adc     temp3,temp0
                sts     RS232LengthPosPtr,temp2                 ;save length of buffer RS232 code
                sts     RS232LengthPosPtr+1,temp3               ;save length of buffer RS232 code
                pop     RS232BufptrXH
                pop     RS232BufptrX
 NoIncRS232BufferFull:
                pop     temp3
                pop     temp2
                pop     temp0
                out     SREG,backupSREGTimer            ;restore SREG
                cli                                     ;disable interrupt because to prevent reentrant interrupt call
                sbi     UCR,RXCIE                       ;enable interrupt from receiving of UART
                reti
;------------------------------------------------------------------------------------------
;*********************************************************************
;* Init program
;*********************************************************************
;------------------------------------------------------------------------------------------
reset:                  ;initialization of processor and variables to right values
                ldi     temp0,StackBegin        ;initialization of stack
                out     SPL,temp0

                clr     XH                              ;RS232 pointer
                clr     YH                              ;USB pointer
                clr     ZH                              ;ROM pointer
                ldi     temp0,LOW(RS232FIFOBegin)       ;set Low to begin of buffer
                sts     RS232ReadPosPtr,temp0           ;zero index of reading
                sts     RS232WritePosPtr,temp0          ;zero index of writing
                ldi     temp0,HIGH(RS232FIFOBegin)      ;set High to begin of buffer
                sts     RS232ReadPosPtr+1,temp0         ;zero index of reading
                sts     RS232WritePosPtr+1,temp0        ;zero index of writing
                sts     RS232LengthPosPtr,YH            ;zero index of length
                sts     RS232LengthPosPtr+1,YH          ;zero index of length
                clr     RS232BufferFull


                rcall   InitACKBufffer          ;initialization of ACK buffer
                rcall   InitNAKBufffer          ;initialization of NAK buffer

                rcall   USBReset                ;initialization of USB addresses

                ldi     temp0,0b00111100        ;set pull-up on PORTB
                out     PORTB,temp0
                ldi     temp0,0b11111111        ;set pull-up on PORTC
                out     PORTC,temp0
                ldi     temp0,0b11111011        ;set pull-up on PORTD
                out     PORTD,temp0

                clr     temp0                   ;
                out     UBRRH,temp0             ;set UART speed High
                out     EEARH,temp0             ;zero EEPROM index

                ldi     temp0,1<<U2X            ;set mode X2 on UART
                out     USR,temp0
                ldi     temp0,InitBaudRate      ;set UART speed
                out     UBRR,temp0
                sbi     UCR,TXEN                ;enable transmiting of UART
                sbi     UCR,RXEN                ;enable receiving of UART
                sbi     UCR,RXCIE               ;enable interrupt from receiving of UART

                ldi     temp0,0x0F              ;INT0 - respond to leading edge
                out     MCUCR,temp0             ;
                ldi     temp0,1<<INT0           ;enable external interrupt INT0
                out     GIMSK,temp0
;------------------------------------------------------------------------------------------
;*********************************************************************
;* Main program
;*********************************************************************
                sei                                     ;enable interrupts globally
Main:
                sbis    inputport,DATAminus     ;waiting till change D- to 0
                rjmp    CheckUSBReset           ;and check, if isn't USB reset

                cpi     ActionFlag,DoReceiveSetupData
                breq    ProcReceiveSetupData
                cpi     ActionFlag,DoPrepareOutContinuousBuffer
                breq    ProcPrepareOutContinuousBuffer
                rjmp    Main

CheckUSBReset:
                ldi     temp0,255               ;counter duration of reset (according to specification is that cca 10ms - here is cca 100us)
WaitForUSBReset:
                sbic    inputport,DATAminus     ;waiting till change D+ to 0
                rjmp    Main
                dec     temp0
                brne    WaitForUSBReset
                rcall   USBReset
                rjmp    Main

ProcPrepareOutContinuousBuffer:
                rcall   PrepareOutContinuousBuffer      ;prepare next sequence of answer to buffer
                ldi     ActionFlag,DoReadySendAnswer
                rjmp    Main
ProcReceiveSetupData:
                ldi     USBBufptrY,InputBufferBegin     ;pointer to begin of receiving buffer
                mov     ByteCount,InputBufferLength     ;length of input buffer
                rcall   DecodeNRZI              ;transfer NRZI coding to bits
                rcall   MirrorInBufferBytes     ;invert bits order in bytes
                rcall   BitStuff                ;removal of bitstuffing
                ;rcall  CheckCRCIn              ;check CRC
                rcall   PrepareUSBOutAnswer     ;prepare answers to transmitting buffer
                ldi     ActionFlag,DoReadySendAnswer
                rjmp    Main
;*********************************************************************
;* Main program END
;*********************************************************************
;------------------------------------------------------------------------------------------
;*********************************************************************
;* Interrupt0 interrupt handler
;*********************************************************************
INT0Handler:                            ;interrupt INT0
                in      backupSREG,SREG
                push    temp0
                push    temp1

                ldi     temp0,3                 ;counter of duration log0
                ldi     temp1,2                 ;counter of duration log1
                ;waiting for begin packet
CheckchangeMinus:
                sbis    inputport,DATAminus     ;waiting till change D- to 1
                rjmp    CheckchangeMinus
CheckchangePlus:
                sbis    inputport,DATAplus      ;waiting till change D+ to 1
                rjmp    CheckchangePlus
DetectSOPEnd:
                sbis    inputport,DATAplus
                rjmp    Increment0              ;D+ =0
Increment1:
                ldi     temp0,3                 ;counter of duration log0
                dec     temp1                   ;how many cycles takes log1
                nop
                breq    USBBeginPacket          ;if this is end of SOP - receive packet
                rjmp    DetectSOPEnd
Increment0:
                ldi     temp1,2                 ;counter of duration log1
                dec     temp0                   ;how many cycles take log0
                nop
                brne    DetectSOPEnd            ;if there isn't SOF - continue
                rjmp    EndInt0HandlerPOP2
EndInt0Handler:
                pop     ACC
                pop     RS232BufptrX
                pop     temp3
                pop     temp2
EndInt0HandlerPOP:
                pop     USBBufptrY
                pop     ByteCount
                mov     bitcount,backupbitcount ;restore bitcount register
EndInt0HandlerPOP2:
                pop     temp1
                pop     temp0
                out     SREG,backupSREG
                ldi     shiftbuf,1<<INTF0       ;zero interruptu flag INTF0
                out     GIFR,shiftbuf
                reti                            ;otherwise finish (was only SOF - every millisecond)

USBBeginPacket:
                mov     backupbitcount,bitcount ;backup bitcount register
                in      shiftbuf,inputport      ;if yes load it as zero bit directly to shift register
USBloopBegin:
                push    ByteCount               ;additional backup of registers (save of time)
                push    USBBufptrY
                ldi     bitcount,6              ;initialization of bits counter in byte
                ldi     ByteCount,MAXUSBBYTES   ;initialization of max number of received bytes in packet
                ldi     USBBufptrY,InputShiftBufferBegin        ;set the input buffer
USBloop1_6:
                in      inputbuf,inputport
                cbr     inputbuf,USBpinmask     ;unmask low 2 bits
                breq    USBloopEnd              ;if they are zeros - end of USB packet
                ror     inputbuf                ;transfer Data+ to shift register
                rol     shiftbuf
                dec     bitcount                ;decrement bits counter
                brne    USBloop1_6              ;if it isn't zero - repeat filling of shift register
                nop                             ;otherwise is necessary copy shift register to buffer
USBloop7:
                in      inputbuf,inputport
                cbr     inputbuf,USBpinmask     ;unmask low 2 bits
                breq    USBloopEnd              ;if they are zeros - end of USB packet
                ror     inputbuf                ;transfer Data+ to shift register
                rol     shiftbuf
                ldi     bitcount,7              ;initialization of bits counter in byte
                st      Y+,shiftbuf             ;copy shift register into buffer and increment pointer to buffer
USBloop0:                                       ;and start receiving next byte
                in      shiftbuf,inputport      ;zero bit directly to shift register
                cbr     shiftbuf,USBpinmask     ;unmask low 2 bits
                breq    USBloopEnd              ;if they are zeros - end of USB packet
                dec     bitcount                ;decrement bits counter
                nop                             ;
                dec     ByteCount               ;if not reached maximum buffer
                brne    USBloop1_6              ;then receive next

                rjmp    EndInt0HandlerPOP       ;otherwise repeat back from begin

USBloopEnd:
                cpi     USBBufptrY,InputShiftBufferBegin+3      ;if at least 3 byte not received
                brcs    EndInt0HandlerPOP       ;then finish
                lds     temp0,InputShiftBufferBegin+0   ;identifier of packet to temp0
                lds     temp1,InputShiftBufferBegin+1   ;address to temp1
                brne    TestDataPacket          ;if is length different from 3 - then this can be only DataPaket
TestIOPacket:
;               cp      temp1,MyAddress         ;if this isn't assigned (address) for me
;               brne    TestDataPacket          ;then this can be still DataPacket
TestSetupPacket:;test to SETUP packet
                cpi     temp0,nNRZISETUPPID
                brne    TestOutPacket           ;if this isn't Setup PID - decode other packet
                cp      temp1,MyInAddress       ;if this isn't assigned (address) for me
                brne    TestDataPacket          ;then this can be still DataPacket
                ldi     State,SetupState
                rjmp    EndInt0HandlerPOP       ;if this is Setup PID - receive consecutive Data packet
TestOutPacket:  ;test for OUT packet
                cpi     temp0,nNRZIOUTPID
                brne    TestInPacket            ;if this isn't Out PID - decode other packet
                cp      temp1,MyOutAddress      ;if this isn't assigned (address) for me
                brne    TestDataPacket          ;then this can be still DataPacket
                ldi     State,OutState
                rjmp    EndInt0HandlerPOP       ;if this is Out PID - receive consecutive Data packet
TestInPacket:   ;test on IN packet
                cpi     temp0,nNRZIINPID
                brne    TestDataPacket          ;if this isn't In PID - decode other packet
                cp      temp1,MyInAddress       ;if this isn't assigned (address) for me
                breq    AnswerToInRequest
TestDataPacket: ;test for DATA0 and DATA1 packet
                cpi     temp0,nNRZIDATA0PID
                breq    Data0Packet             ;if this isn't Data0 PID - decode other packet
                cpi     temp0,nNRZIDATA1PID
                brne    NoMyPacked              ;if this isn't Data1 PID - decode other packet
Data0Packet:
```

```
                cpi     State,SetupState        ;if was state Setup
                breq    ReceiveSetupData        ;receive it
                cpi     State,OutState          ;if was state Out
                breq    ReceiveOutData          ;receive it
NoMyPacked:
                ldi     State,BaseState         ;zero state
                rjmp    EndInt0HandlerPOP       ;and receive consecutive Data packet

AnswerToInRequest:
                push    temp2                   ;backup next registers and continue
                push    temp3
                push    RS232BufptrX
                push    ACC
                cpi     ActionFlag,DoReadySendAnswer    ;if isn't prepared answer
                brne    NoReadySend             ;then send NAK
                rcall   SendPreparedUSBAnswer   ;transmitting answer back
                cpi     State,AddressChangeState ;if state is AddressChange
                breq    SetMyNewUSBAddress      ;then is necessary to change USB address
                ldi     State,InState
                ldi     ActionFlag,DoPrepareOutContinuousBuffer
                rjmp    EndInt0Handler          ;and repeat - wait for next response from USB
ReceiveSetupData:
                push    temp2                   ;backup next registers and continue
                push    temp3
                push    RS232BufptrX
                push    ACC
                rcall   SendACK                 ;accept Setup Data packet
                rcall   FinishReceiving         ;finish receiving
                ldi     ActionFlag,DoReceiveSetupData
                rjmp    EndInt0Handler
ReceiveOutData:
                push    temp2                   ;backup next registers and continue
                push    temp3
                push    RS232BufptrX
                push    ACC
                cpi     ActionFlag,DoReceiveSetupData   ;if is currently in process command Setup
                breq    NoReadySend             ;then send NAK
                rcall   SendACK                 ;accept Out packet
                clr     ActionFlag
                rjmp    EndInt0Handler
NoReadySend:
                rcall   SendNAK                 ;still I am not ready to answer
                rjmp    EndInt0Handler          ;and repeat - wait for next response from USB
;------------------------------------------------------------------------------------------
SetMyNewUSBAddress:                     ;set new USB address in NRZI coded
                lds     MyInAddress,MyInAddressSRAM
                lds     MyOutAddress,MyOutAddressSRAM
                rjmp    EndInt0Handler
;------------------------------------------------------------------------------------------
FinishReceiving:                        ;corrective actions for receive termination
                cpi     bitcount,7              ;transfer to buffer also last not completed byte
                breq    NoRemainingBits         ;if were all bytes transfered, then nothing transfer
                inc     bitcount
ShiftRemainingBits:
                rol     shiftbuf                ;shift remaining not completed bits on right position
                dec     bitcount
                brne    ShiftRemainingBits
                st      Y+,shiftbuf             ;and copy shift register bo buffer - not completed byte
NoRemainingBits:
                mov     ByteCount,USBBufptrY
                subi    ByteCount,InputShiftBufferBegin-1       ;in ByteCount is number of received bytes (including not completed bytes)

                mov     InputBufferLength,ByteCount             ;and save for use in main program
                ldi     USBBufptrY,InputShiftBufferBegin        ;pointer to begin of receiving shift buffer
                ldi     RS232BufptrX,InputBufferBegin+1         ;data buffer (leave out SOP)
                push    XH                                      ;save RS232BufptrX Hi index
                clr     XH
MoveDataBuffer:
                ld      temp0,Y+
                st      X+,temp0
                dec     ByteCount
                brne    MoveDataBuffer

                pop     XH                                      ;restore RS232BufptrX Hi index
                ldi     ByteCount,nNRZISOPbyte
                sts     InputBufferBegin,ByteCount              ;like received SOP - it is not copied from shift buffer
                ret
;------------------------------------------------------------------------------------------
;*********************************************************************
;* Other procedures
;*********************************************************************
;------------------------------------------------------------------------------------------
USBReset:               ;initialization of USB state engine
                ldi     temp0,nNRZIADDR0        ;initialization of USB address
                mov     MyOutAddress,temp0
                mov     MyInAddress,temp0
                clr     State                   ;initialization of state engine
                clr     BitStuffInOut
                clr     OutBitStuffNumber
                clr     ActionFlag
                clr     RAMread                 ;will be reading from ROM
                sts     ConfigByte,RAMread      ;unconfigured state
                ret
;------------------------------------------------------------------------------------------
SendPreparedUSBAnswer:  ;transmitting by NRZI coding OUT buffer with length OutputBufferLength to USB
                mov     ByteCount,OutputBufferLength            ;length of answer
SendUSBAnswer:  ;transmitting by NRZI coding OUT buffer to USB
                ldi     USBBufptrY,OutputBufferBegin            ;pointer to begin of transmitting buffer
SendUSBBuffer:  ;transmitting by NRZI coding given buffer to USB
                ldi     temp1,0                 ;incrementing pointer (temporary variable)
                mov     temp3,ByteCount         ;byte counter: temp3 = ByteCount
                ldi     temp2,0b00000011        ;mask for xoring
                ld      inputbuf,Y+             ;load first byte to inputbuf and increment pointer to buffer
                                                ;USB as output:
                cbi     outputport,DATAplus     ;down DATAPLUS : idle state of USB port
                sbi     outputport,DATAminus    ;set DATAMINUS : idle state of USB port
                sbi     USBdirection,DATAplus   ;DATAPLUS as output
                sbi     USBdirection,DATAminus  ;DATAMINUS as output

                in      temp0,outputport        ;idle state of USB port to temp0
SendUSBAnswerLoop:
                ldi     bitcount,7              ;bits counter
```

```
SendUSBAnswerByteLoop:
                nop                             ;delay because timing
                ror     inputbuf                ;to carry transmiting bit (in direction first LSB then MSB)
                brcs    NoXORSend               ;if that it is one - don't change USB state
                eor     temp0,temp2             ;otherwise state will be changed
NoXORSend:
                out     outputport,temp0        ;send out to USB
                dec     bitcount                ;decrement bits counter - according to carry flag
                brne    SendUSBAnswerByteLoop   ;if bits counter isn't zero - repeat transmiting with next bit
                sbrs    inputbuf,0              ;if is transmiting bit one - don't change USB state
                eor     temp0,temp2             ;otherwise state will be changed
NoXORSendLSB:
                dec     temp3                   ;decrement bytes counter
                ld      inputbuf,Y+             ;load next byte and increment pointer to buffer
                out     outputport,temp0        ;transmit to USB
                brne    SendUSBAnswerLoop       ;repeat for all buffer (till temp3=0)

                mov     bitcount,OutBitStuffNumber      ;bits counter for bitstuff
                cpi     bitcount,0              ;if not be needed bitstuff
                breq    ZeroBitStuf
SendUSBAnswerBitstuffLoop:
                ror     inputbuf                ;to carry transmiting bit (in direction first LSB then MSB)
                brcs    NoXORBitstuffSend       ;if is one - don't change state on USB
                eor     temp0,temp2             ;otherwise state will be changed
NoXORBitstuffSend:
                out     outputport,temp0        ;transmit to USB
                nop                             ;delay because of timing
                dec     bitcount                ;decrement bits counter - according to carry flag
                brne    SendUSBAnswerBitstuffLoop       ;if bits counter isn't zero - repeat transmiting with next bit
                ld      inputbuf,Y              ;delay 2 cycle
ZeroBitStuf:
                nop                             ;delay 1 cycle
                cbr     temp0,3
                out     outputport,temp0        ;transmit EOP on USB

                ldi     bitcount,5              ;delay counter: EOP shouls exists 2 bits (16 cycle at 12MHz)
SendUSBWaitEOP:
                dec     bitcount
                brne    SendUSBWaitEOP

                sbi     outputport,DATAminus    ;set DATAMINUS : idle state on USB port
                sbi     outputport,DATAminus    ;delay 2 cycle: Idle should exists 1 bit (8 cycle at 12MHz)
                cbi     USBdirection,DATAplus   ;DATAPLUS as input
                cbi     USBdirection,DATAminus  ;DATAMINUS as input
                cbi     outputport,DATAminus    ;reset DATAMINUS : the third state on USB port
                ret
;------------------------------------------------------------------------------------------
ToggleDATAPID:
                lds     temp0,OutputBufferBegin+1       ;load last PID
                cpi     temp0,DATA1PID                  ;if last was DATA1PID byte
                ldi     temp0,DATA0PID
                breq    SendData0PID                    ;then send zero answer with DATA0PID
                ldi     temp0,DATA1PID                  ;otherwise send zero answer with DATA1PID
SendData0PID:
                sts     OutputBufferBegin+1,temp0       ;DATA0PID byte
                ret
```

```
;------------------------------------------------------------------------------------------
ComposeZeroDATA1PIDAnswer:
                ldi     temp0,DATA0PID                  ;DATA0 PID - in the next will be toggled to DATA1PID in load descriptor
                sts     OutputBufferBegin+1,temp0       ;load to output buffer
ComposeZeroAnswer:
                ldi     temp0,SOPbyte
                sts     OutputBufferBegin+0,temp0       ;SOP byte
                rcall   ToggleDATAPID                   ;change DATAPID
                ldi     temp0,0x00
                sts     OutputBufferBegin+2,temp0       ;CRC byte
                sts     OutputBufferBegin+3,temp0       ;CRC byte
                ldi     ByteCount,2+2                   ;length of output buffer (SOP and PID + CRC16)
                ret
;------------------------------------------------------------------------------------------
InitACKBufffer:
                ldi     temp0,SOPbyte
                sts     ACKBufferBegin+0,temp0          ;SOP byte
                ldi     temp0,ACKPID
                sts     ACKBufferBegin+1,temp0          ;ACKPID byte
                ret
;------------------------------------------------------------------------------------------
SendACK:
                push    USBBufptrY
                push    bitcount
                push    OutBitStuffNumber
                ldi     USBBufptrY,ACKBufferBegin       ;pointer to begin of ACK buffer
                ldi     ByteCount,2                     ;number of transmit bytes (only SOP and ACKPID)
                clr     OutBitStuffNumber
                rcall   SendUSBBuffer
                pop     OutBitStuffNumber
                pop     bitcount
                pop     USBBufptrY
                ret
;------------------------------------------------------------------------------------------
InitNAKBufffer:
                ldi     temp0,SOPbyte
                sts     NAKBufferBegin+0,temp0          ;SOP byte
                ldi     temp0,NAKPID
                sts     NAKBufferBegin+1,temp0          ;NAKPID byte
                ret
;------------------------------------------------------------------------------------------
SendNAK:
                push    OutBitStuffNumber
                ldi     USBBufptrY,NAKBufferBegin       ;pointer to begin of ACK buffer
                ldi     ByteCount,2                     ;number of transmited bytes (only SOP and NAKPID)
                clr     OutBitStuffNumber
                rcall   SendUSBBuffer
                pop     OutBitStuffNumber
                ret
;------------------------------------------------------------------------------------------
ComposeSTALL:
                ldi     temp0,SOPbyte
                sts     OutputBufferBegin+0,temp0       ;SOP byte
                ldi     temp0,STALLPID
                sts     OutputBufferBegin+1,temp0       ;STALLPID byte
                ldi     ByteCount,2                     ;length of output buffer (SOP and PID)
                ret
;------------------------------------------------------------------------------------------
DecodeNRZI:     ;encoding of buffer from NRZI code to binary
                push    USBBufptrY              ;back up pointer to buffer
                push    ByteCount               ;back up length of buffer
                add     ByteCount,USBBufptrY    ;end of buffer to ByteCount
                ser     temp0                   ;to ensure unit carry (in the next rotation)
NRZIloop:
                ror     temp0                   ;filling carry from previous byte
                ld      temp0,Y                 ;load received byte from buffer
                mov     temp2,temp0             ;shifted register to one bit to the right and XOR for function of NRZI decoding
                ror     temp2                   ;carry to most significant digit bit and shift
                eor     temp2,temp0             ;NRZI decoding
                com     temp2                   ;negate
                st      Y+,temp2                ;save back as decoded byte and increment pointer to buffer
                cp      USBBufptrY,ByteCount    ;if not all bytes
                brne    NRZIloop                ;then repeat
                pop     ByteCount               ;restore buffer length
                pop     USBBufptrY              ;restore pointer to buffer
                ret                             ;otherwise finish
;------------------------------------------------------------------------------------------
BitStuff:       ;removal of bitstuffing in buffer
                clr     temp3                   ;counter of omitted bits
                clr     lastBitstufNumber       ;0xFF to lastBitstufNumber
                dec     lastBitstufNumber
BitStuffRepeat:
                push    USBBufptrY              ;back up pointer to buffer
                push    ByteCount               ;back up buffer length
                mov     temp1,temp3             ;counter of all bits
                ldi     temp0,8                 ;sum all bits in buffer
SumAllBits:
                add     temp1,temp0
                dec     ByteCount
                brne    SumAllBits
                ldi     temp2,6                 ;initialize counter of ones
                pop     ByteCount               ;restore buffer length
                push    ByteCount               ;back up buffer length
                add     ByteCount,USBBufptrY    ;end of buffer to ByteCount
                inc     ByteCount               ;and for safety increment it with 2 (because of shifting)
                inc     ByteCount
BitStuffLoop:
                ld      temp0,Y                 ;load received byte from buffer
                ldi     bitcount,8              ;bits counter in byte
BitStuffByteLoop:
                ror     temp0                   ;filling carry from LSB
                brcs    IncrementBitstuff       ;if that LSB=0
                ldi     temp2,7                 ;initialize counter of ones +1 (if was zero)
IncrementBitstuff:
                dec     temp2                   ;decrement counter of ones (assumption of one bit)
                brne    DontShiftBuffer         ;if there was not 6 ones together - don't shift buffer
                cp      temp1,lastBitstufNumber ;
                ldi     temp2,6                 ;initialize counter of ones (if no bitstuffing will be made then must be started again)
                brcc    DontShiftBuffer         ;if already was made bitstuffing - don't shift buffer

                dec     temp1   ;
                mov     lastBitstufNumber,temp1 ;remember last position of bitstuffing
                cpi     bitcount,1              ;for pointing to 7-th bit (which must be deleted or where to insert zero)
                brne    NoBitcountCorrect
                ldi     bitcount,9      ;
                inc     USBBufptrY              ;zvys pointer do buffera        ENG;increment pointer to buffer
NoBitcountCorrect:
                dec     bitcount
                bst     BitStuffInOut,0
                brts    CorrectOutBuffer        ;if this is Out buffer - increment buffer length
                rcall   ShiftDeleteBuffer       ;shift In buffer
                dec     temp3                   ;decrement counter of omission
                rjmp    CorrectBufferEnd
CorrectOutBuffer:
                rcall   ShiftInsertBuffer       ;shift Out buffer
                inc     temp3                   ;increment counter of omission
CorrectBufferEnd:
                pop     ByteCount               ;restore buffer length
                pop     USBBufptrY              ;restore pointer to buffer
                rjmp    BitStuffRepeat          ;and restart from begin
DontShiftBuffer:
                dec     temp1                   ;if already were all bits
                breq    EndBitStuff             ;finish cycle
                dec     bitcount                ;decrement bits counter in byte
                brne    BitStuffByteLoop        ;if not yet been all bits in byte - go to next bit
                                                ;otherwise load next byte
                inc     USBBufptrY              ;increment pointer to buffer
                rjmp    BitStuffLoop            ;and repeat
EndBitStuff:
                pop     ByteCount               ;restore buffer length
                pop     USBBufptrY              ;restore pointer to buffer
                bst     BitStuffInOut,0
                brts    IncrementLength         ;if this is Out buffer - increment length of Out buffer
DecrementLength:                                ;if this is In buffer - decrement length of In buffer
                cpi     temp3,0                 ;was at least one decrement
                breq    NoChangeByteCount       ;if no - don't change buffer length
                dec     ByteCount               ;if this is In buffer - decrement buffer length
                subi    temp3,256-8             ;if there wasn't above 8 bits over
                brcc    NoChangeByteCount       ;then finish
                dec     ByteCount               ;otherwise next decrement buffer length
                ret                             ;and finish
IncrementLength:
                mov     OutBitStuffNumber,temp3 ;remember number of bits over
                subi    temp3,8                 ;if there wasn't above 8 bits over
                brcs    NoChangeByteCount       ;then finish
                inc     ByteCount               ;otherwise increment buffer length
                mov     OutBitStuffNumber,temp3 ;and remember number of bits over (decremented by 8)
NoChangeByteCount:
                ret                             ;finish
;------------------------------------------------------------------------------------------
ShiftInsertBuffer:      ;shift buffer by one bit to right from end till to position: byte-USBBufptrY and bit-bitcount
                mov     temp0,bitcount          ;calculation: bitcount= 9-bitcount
                ldi     bitcount,9
                sub     bitcount,temp0          ;to bitcount bit position, which is necessary to clear

                ld      temp1,Y                 ;load byte which still must be shifted from position bitcount
                rol     temp1                   ;and shift to the left through Carry (transmission from higher byte and LSB to Carry)
                ser     temp2                   ;FF to mask - temp2
```

```
HalfInsertPosuvMask:
                lsl     temp2                   ;zero to the next low bit of mask
                dec     bitcount                ;till not reached boundary of shifting in byte
                brne    HalfInsertPosuvMask

                and     temp1,temp2             ;unmask that remains only high shifted bits in temp1
                com     temp2                   ;invert mask
                lsr     temp2                   ;shift mask to the right - for insertion of zero bit
                ld      temp0,Y                 ;load byte which must be shifted from position bitcount to temp0
                and     temp0,temp2             ;unmask to remains only low non-shifted bits in temp0
                or      temp1,temp0             ;and put together shifted and nonshifted part

                ld      temp0,Y                 ;load byte which must be shifted from position bitcount
                rol     temp0                   ;and shift it to the left through Carry (to set right Carry for further carry)
                st      Y+,temp1                ;and load back modified byte
ShiftInsertBufferLoop:
                cpse    USBBufptrY,ByteCount    ;if are not all entire bytes
                rjmp    NoEndShiftInsertBuffer  ;then continue
                ret                             ;otherwise finish
NoEndShiftInsertBuffer:
                ld      temp1,Y                 ;load byte
                rol     temp1                   ;and shift to the left through Carry (carry from low byte and LSB to Carry)
                st      Y+,temp1                ;and store back
                rjmp    ShiftInsertBufferLoop   ;and continue
;------------------------------------------------------------------------------------------
ShiftDeleteBuffer:      ;shift buffer one bit to the left from end to position: byte-USBBufptrY and bit-bitcount
                mov     temp0,bitcount          ;calculation: bitcount= 9-bitcount
                ldi     bitcount,9
                sub     bitcount,temp0          ;to bitcount bit position, which must be shifted
                mov     temp0,USBBufptrY        ;backup pointera to buffer
                inc     temp0                   ;position of completed bytes to temp0
                mov     USBBufptrY,ByteCount    ;maximum position to pointer
ShiftDeleteBufferLoop:
                ld      temp1,-Y                ;decrement buffer and load byte
                ror     temp1                   ;and right shift through Carry (carry from higher byte and LSB to Carry)
                st      Y,temp1                 ;and store back
                cpse    USBBufptrY,temp0        ;if there are not all entire bytes
                rjmp    ShiftDeleteBufferLoop   ;then continue

                ld      temp1,-Y                ;decrement buffer and load byte which must be shifted from position bitcount
                ror     temp1                   ;and right shift through Carry (carry from higher byte and LSB to Carry)
                ser     temp2                   ;FF to mask - temp2
HalfDeletePosuvMask:
                dec     bitcount                ;till not reached boundary of shifting in byte
                breq    DoneMask
                lsl     temp2                   ;zero to the next low bit of mask
                rjmp    HalfDeletePosuvMask
DoneMask:
                and     temp1,temp2             ;unmask to remain only high shifted bits in temp1
                com     temp2                   ;invert mask
                ld      temp0,Y                 ;load byte which must be shifted from position bitcount to temp0
                and     temp0,temp2             ;unmask to remain only low nonshifted bits in temp0
                or      temp1,temp0             ;and put together shifted and nonshifted part
                st      Y,temp1                 ;and store back
                ret                             ;and finish
;------------------------------------------------------------------------------------------
```

```
MirrorInBufferBytes:
                push    USBBufptrY
                push    ByteCount
                ldi     USBBufptrY,InputBufferBegin
                rcall   MirrorBufferBytes
                pop     ByteCount
                pop     USBBufptrY
                ret
;------------------------------------------------------------------------------------------
MirrorBufferBytes:
                add     ByteCount,USBBufptrY    ;ByteCount shows to the end of message
MirrorBufferloop:
                ld      temp0,Y                 ;load received byte from buffer
                ldi     temp1,8                 ;bits counter
MirrorBufferByteLoop:
                ror     temp0                   ;to carry next least bit
                rol     temp2                   ;from carry next bit to reverse order
                dec     temp1                   ;was already entire byte
                brne    MirrorBufferByteLoop    ;if no then repeat next least bit
                st      Y+,temp2                ;save back as reversed byte  and increment pointer to buffer
                cp      USBBufptrY,ByteCount    ;if not yet been all
                brne    MirrorBufferloop        ;then repeat
                ret                             ;otherwise finish
;------------------------------------------------------------------------------------------
;CheckCRCIn:
;               kiss    USBBUFPTRY
;               kiss    ByteCount
;               ldi     USBBUFPTRY,InputBuffercompare
;               rcall   CheckCRC
;               pope    ByteCount
;               pope    USBBUFPTRY
;               lip
;------------------------------------------------------------------------------------------
AddCRCOut:
                push    USBBufptrY
                push    ByteCount
                ldi     USBBufptrY,OutputBufferBegin
                rcall   CheckCRC
                com     temp0                   ;negation of CRC
                com     temp1
                st      Y+,temp1                ;save CRC to the end of buffer (at first MSB)
                st      Y,temp0                 ;save CRC to the end of buffer (then LSB)
                dec     USBBufptrY              ;pointer to CRC position
                ldi     ByteCount,2             ;reverse bits order in 2 bytes CRC
                rcall   MirrorBufferBytes       ;reverse bits order in CRC (transmiting CRC - MSB first)
                pop     ByteCount
                pop     USBBufptrY
                ret
;------------------------------------------------------------------------------------------
CheckCRC:       ;input: USBBufptrY = begin of message   ,ByteCount = length of message
                add     ByteCount,USBBufptrY    ;ByteCount points to the end of message
                inc     USBBufptrY              ;set the pointer to message start - omit SOP
                ld      temp0,Y+                ;load PID to temp0
                                                ;and set the pointer to start of message - omit also PID
                cpi     temp0,DATA0PID          ;if is DATA0 field
                breq    ComputeDATACRC          ;compute CRC16
                cpi     temp0,DATA1PID          ;if is DATA1 field
                brne    CRC16End                ;if no then finish
ComputeDATACRC:
                ser     temp0                   ;initialization of remaider LSB to 0xff
                ser     temp1                   ;initialization of remaider MSB to 0xff
CRC16Loop:
                ld      temp2,Y+                ;load message to temp2 and increment pointer to buffer
                ldi     temp3,8                 ;bits counter in byte - temp3
CRC16LoopByte:
                bst     temp1,7                 ;to T save MSB of remainder (remainder is only 16 bits - 8 bit of higher byte)
                bld     bitcount,0              ;to bitcount LSB save T - of MSB remainder
                eor     bitcount,temp2          ;XOR of bit message and bit remainder - in LSB bitcount
                rol     temp0                   ;shift remainder to the left - low byte (two bytes - through carry)
                rol     temp1                   ;shift remainder to the left - high byte (two bytes - through carry)
                cbr     temp0,1                 ;znuluj LSB remains
                lsr     temp2                   ;shift message to right
                ror     bitcount                ;result of XOR bits from LSB to carry
                brcc    CRC16NoXOR              ;if is XOR bitmessage and MSB of remainder = 0 , then no XOR
                ldi     bitcount,CRC16poly>>8   ;to bitcount CRC polynomial - high byte
                eor     temp1,bitcount          ;and make XOR from remains and CRC polynomial - high byte
                ldi     bitcount,CRC16poly      ;to bitcount CRC polynomial - low byte
                eor     temp0,bitcount          ;and make XOR of remainder and CRC polynomial - low byte
CRC16NoXOR:
                dec     temp3                   ;were already all bits in byte
                brne    CRC16LoopByte           ;unless, then go to next bit
                cp      USBBufptrY,ByteCount    ;was already end-of-message
                brne    CRC16Loop               ;unless then repeat
CRC16End:
                ret                             ;otherwise finish (in temp0 and temp1 is result)
;------------------------------------------------------------------------------------------
LoadDescriptorFromROM:
                lpm                             ;load from ROM position pointer to R0
                st      Y+,R0                   ;R0 save to buffer and increment buffer
                adiw    ZH:ZL,1                 ;increment index to ROM
                dec     ByteCount               ;till are not all bytes
                brne    LoadDescriptorFromROM   ;then load next
                rjmp    EndFromRAMROM           ;otherwise finish
;------------------------------------------------------------------------------------------
LoadDescriptorFromROMZeroInsert:
                lpm                             ;load from ROM position pointer to R0
                st      Y+,R0                   ;R0 save to buffer and increment buffer

                bst     RAMread,3               ;if bit 3 is one - don't insert zero
                brtc    InsertingZero           ;otherwise zero will be inserted
                adiw    ZH:ZL,1                 ;increment index to ROM
                lpm                             ;load from ROM position pointer to R0
                st      Y+,R0                   ;R0 save to buffer and increment buffer
                clt                             ;and clear
                bld     RAMread,3               ;the third bit in RAMread - for to the next zero insertion will be made
                rjmp    InsertingZeroEnd        ;and continue
InsertingZero:
                clr     R0                      ;for insertion of zero
                st      Y+,R0                   ;zero save to buffer and increment buffer
InsertingZeroEnd:
                adiw    ZH:ZL,1                 ;increment index to ROM
                subi    ByteCount,2             ;till are not all bytes
                brne    LoadDescriptorFromROMZeroInsert ;then load next
                rjmp    EndFromRAMROM           ;otherwise finish
;------------------------------------------------------------------------------
LoadDescriptorFromSRAM:
                ld      R0,Z                    ;load from position RAM pointer to R0
                st      Y+,R0                   ;R0 save to buffer and increment buffer
                adiw    ZH:ZL,1                 ;increment index to RAM
                dec     ByteCount               ;till are not all bytes
                brne    LoadDescriptorFromSRAM  ;then load next
                rjmp    EndFromRAMROM           ;otherwise finish
;------------------------------------------------------------------------------
LoadDescriptorFromEEPROM:
                out     EEARL,ZL                ;set the address EEPROM Lo
                out     EEARH,ZH                ;set the address EEPROM Hi
                sbi     EECR,EERE               ;read EEPROM to register EEDR
                in      R0,EEDR                 ;load from EEDR to R0
                st      Y+,R0                   ;R0 save to buffer and increment buffer
                adiw    ZH:ZL,1                 ;increment index to EEPROM
                dec     ByteCount               ;till are not all bytes
                brne    LoadDescriptorFromEEPROM;then load next
                rjmp    EndFromRAMROM           ;otherwise finish
;------------------------------------------------------------------------------
LoadXXXDescriptor:
                ldi     temp0,SOPbyte                   ;SOP byte
                sts     OutputBufferBegin,temp0         ;to begin of tramsmiting buffer store SOP
                ldi     ByteCount,8                     ;8 byte store
                ldi     USBBufptrY,OutputBufferBegin+2  ;to transmitting buffer

                and     RAMread,RAMread                 ;if will be reading from RAM or ROM or EEPROM
                brne    FromRAMorEEPROM                 ;0=ROM,1=RAM,2=EEPROM,4=ROM with zero insertion (string)
FromROM:
                rjmp    LoadDescriptorFromROM           ;load descriptor from ROM
FromRAMorEEPROM:
                sbrc    RAMread,2                       ;if RAMREAD=4
                rjmp    LoadDescriptorFromROMZeroInsert ;read from ROM with zero insertion
                sbrc    RAMread,0                       ;if RAMREAD=1
                rjmp    LoadDescriptorFromSRAM          ;load data from SRAM
                rjmp    LoadDescriptorFromEEPROM        ;otherwise read from EEPROM
EndFromRAMROM:
                sbrc    RAMread,7                       ;if is most significant bit in variable RAMread=1
                clr     RAMread                         ;clear RAMread
                rcall   ToggleDATAPID                   ;change DATAPID
                ldi     USBBufptrY,OutputBufferBegin+1  ;to transmitting buffer - position of DATA PID
                ret
;------------------------------------------------------------------------------
PrepareUSBOutAnswer:    ;prepare answer to buffer
                rcall   PrepareUSBAnswer                ;prepare answer to buffer
MakeOutBitStuff:
                inc     BitStuffInOut                   ;transmitting buffer - insertion of bitstuff bits
                ldi     USBBufptrY,OutputBufferBegin    ;to transmitting buffer
                rcall   BitStuff
                mov     OutputBufferLength,ByteCount    ;length of answer store for transmiting
                clr     BitStuffInOut                   ;receiving buffer - deletion of bitstuff bits
                ret
;------------------------------------------------------------------------------
PrepareUSBAnswer:       ;prepare answer to buffer
```

```
                clr     RAMread                         ;zero to RAMread variable - reading from ROM
                lds     temp0,InputBufferBegin+2        ;bmRequestType to temp0
                lds     temp1,InputBufferBegin+3        ;bRequest to temp1
                cbr     temp0,0b10011111                ;if is 5 and 6 bit zero
                brne    VendorRequest                   ;then this isn't  Vendor Request
                rjmp    StandardRequest                 ;but this is standard Request
;--------------------------
VendorRequest:
                clr     ZH                              ;for reading from RAM or EEPROM

                cpi     temp1,1                         ;
                brne    NoDoSetInfraBufferEmpty         ;
                rjmp    DoSetInfraBufferEmpty           ;restart infra receiving (if it was stopped by reading from RAM)
NoDoSetInfraBufferEmpty:
                cpi     temp1,2                         ;
                brne    NoDoGetInfraCode
                rjmp    DoGetInfraCode                  ;transmit received infra code (if it is in buffer)
NoDoGetInfraCode:
                cpi     temp1,3                         ;
                brne    NoDoSetDataPortDirection
                rjmp    DoSetDataPortDirection          ;set flow direction of datal bits
NoDoSetDataPortDirection:
                cpi     temp1,4                         ;
                brne    NoDoGetDataPortDirection
                rjmp    DoGetDataPortDirection          ;detect of flow direction of data bits
NoDoGetDataPortDirection:
                cpi     temp1,5                         ;
                brne    NoDoSetOutDataPort
                rjmp    DoSetOutDataPort                ;set data bits (if they are inputs, then pull-ups)
NoDoSetOutDataPort:
                cpi     temp1,6                         ;
                brne    NoDoGetOutDataPort
                rjmp    DoGetOutDataPort                ;detect settings of data out bits (if they are input, then pull-ups)
NoDoGetOutDataPort:
                cpi     temp1,7                         ;
                brne    NoDoGetInDataPort
                rjmp    DoGetInDataPort                 ;return value of input data port
NoDoGetInDataPort:
                cpi     temp1,8                         ;
                brne    NoDoEEPROMRead
                rjmp    DoEEPROMRead                    ;return contents of EEPROM from given address
NoDoEEPROMRead:
                cpi     temp1,9                         ;
                brne    NoDoEEPROMWrite
                rjmp    DoEEPROMWrite                   ;write to EEPROM to given address given data
NoDoEEPROMWrite:
                cpi     temp1,10                        ;
                brne    NoDoRS232Send
                rjmp    DoRS232Send                     ;transmit byte to serial line
NoDoRS232Send:
                cpi     temp1,11                        ;
                brne    NoDoRS232Read
                rjmp    DoRS232Read                     ;returns received byte from serial line
NoDoRS232Read:
                cpi     temp1,12                        ;
                brne    NoDoSetRS232Baud
                rjmp    DoSetRS232Baud                  ;set line speed of of serial line
NoDoSetRS232Baud:
                cpi     temp1,13                        ;
                brne    NoDoGetRS232Baud
                rjmp    DoGetRS232Baud                  ;return line speed of serial line
NoDoGetRS232Baud:
                cpi     temp1,14                        ;
                brne    NoDoGetRS232Buffer
                rjmp    DoGetRS232Buffer                ;return line speed of serial line
NoDoGetRS232Buffer:
                cpi     temp1,15                        ;
                brne    NoDoSetRS232DataBits
                rjmp    DoSetRS232DataBits              ;set line speed of serial line
NoDoSetRS232DataBits:
                cpi     temp1,16                        ;
                brne    NoDoGetRS232DataBits
                rjmp    DoGetRS232DataBits              ;return line speed of serial line
NoDoGetRS232DataBits:
                cpi     temp1,17                        ;
                brne    NoDoSetRS232Parity
                rjmp    DoSetRS232Parity                ;set line speed of serial line
NoDoSetRS232Parity:
                cpi     temp1,18                        ;
                brne    NoDoGetRS232Parity
                rjmp    DoGetRS232Parity                ;return line speed of serial line
NoDoGetRS232Parity:
                cpi     temp1,19                        ;
                brne    NoDoSetRS232StopBits
                rjmp    DoSetRS232StopBits              ;set line speed of serial line
NoDoSetRS232StopBits:
                cpi     temp1,20                        ;
                brne    NoDoGetRS232StopBits
                rjmp    DoGetRS232StopBits              ;return line speed of serial line
NoDoGetRS232StopBits:

                cpi     temp1,USER_FNC_NUMBER+0         ;
                brne    NoDoUserFunction0
                rjmp    DoUserFunction0                 ;execute of user function0
NoDoUserFunction0:
                cpi     temp1,USER_FNC_NUMBER+1         ;
                brne    NoDoUserFunction1
                rjmp    DoUserFunction1                 ;execute of user function1
NoDoUserFunction1:
                cpi     temp1,USER_FNC_NUMBER+2         ;
                brne    NoDoUserFunction2
                rjmp    DoUserFunction2                 ;execute of user function1
NoDoUserFunction2:

                rjmp    ZeroDATA1Answer                 ;if that it was something unknown, then prepare zero answer


;------------------------------ USER FUNCTIONS --------------------------------------

;------------------------------TEMPLATE OF YOUR FUNCTION----------------------------
;------------------ BEGIN: This is template how to write own function --------------

```

```
;free of use are registers:
        ;temp0,temp1,temp2,temp3,ACC,ZH,ZL
        ;registers are destroyed after execution (use push/pop to save content)

;at the end of routine you must correctly set registers:
        ;RAMread - 0=reading from ROM, 1=reading from RAM, 2=reading from EEPROM
        ;temp0 - number of transmitted data bytes
        ;ZH,ZL - pointer to buffer of transmitted data (pointer to ROM/RAM/EEPROM)

;to transmit data (preparing data to buffer) :
        ;to transmit data you must jump to "ComposeEndXXXDescriptor"
        ;to transmit one zero byte you can jump to "OneZeroAnswer"  (commonly used as confirmation of correct processing)
        ;to transmit two zero byte you can jump to "TwoZeroAnswer"  (commonly used as confirmation of error in processing)
        ;for small size (up to 8 bytes) ansver use buffer AnswerArray (see function DoGetOutDataPort:)

DoUserFunctionX:
DoUserFunction0:  ;send byte(s) of RAM starting at position given by first parameter in function
                lds     temp0,InputBufferBegin+4        ;first parameter Lo into temp0
                lds     temp1,InputBufferBegin+5        ;first  parameter Hi into temp1
                ;lds    temp2,InputBufferBegin+6        ;second parameter Lo into temp2
                ;lds    temp3,InputBufferBegin+7        ;second parameter Hi into temp3
                ;lds    ACC,InputBufferBegin+8          ;number of requested bytes from USB host (computer) into ACC

                ;Here add your own code:
                ;-------------------------------------------------------------------
                nop                                     ;example of code - nothing to do
                nop
                nop
                nop
                nop
                ;-------------------------------------------------------------------

                mov     ZL,temp0                        ;will be sending value of RAM - from address stored in temp0 (first parameter Lo of function)
                mov     ZH,temp1                        ;will be sending value of RAM - from address stored in temp1 (first parameter Hi of function)
                inc     RAMread                         ;RAMread=1 - reading from RAM
                ldi     temp0,255                       ;send max number of bytes - 255 bytes are maximum
                rjmp    ComposeEndXXXDescriptor         ;a prepare data
DoUserFunction1:
                rjmp    OneZeroAnswer                   ;only confirm receiving by one zero byte answer
DoUserFunction2:
                rjmp    TwoZeroAnswer                   ;only confirm receiving by two zero bytes answer
;------------------ END: This is template how to write own function ----------------


;------------------------------ USER FUNCTIONS --------------------------------------
;--------------------------
DoSetInfraBufferEmpty:
                rjmp    OneZeroAnswer                   ;acknowledge reception with single zero
;--------------------------
DoGetInfraCode:
                rjmp    OneZeroAnswer                   ;acknowledge reception with single zero
;--------------------------
DoSetDataPortDirection:
                lds     temp1,InputBufferBegin+7        ;fourth parameter - bit mask - which port(s) to change

                lds     temp0,InputBufferBegin+4        ;first parameter - direction of data bits DDRB
                andi    temp0,0b00111100                ;mask unused pins
                sbrc    temp1,0                         ;if bit0 is zero - don't change port state
                out     DDRB,temp0                      ;and update direction of data port

                lds     temp0,InputBufferBegin+5        ;second parameter - direction of data bits DDRC
                sbrc    temp1,1                         ;if bit1 is zero - don't change port state
                out     DDRC,temp0                      ;and update direction of data port

                lds     temp0,InputBufferBegin+6        ;third parameter - direction of data bits DDRD
                andi    temp0,0b11111000                ;mask unused pins
                ori     temp0,0b00000010                ;mask unused pins
                sbrc    temp1,2                         ;if bit2 is zero - don't change port state
                out     DDRD,temp0                      ;and update direction of data port

                rjmp    OneZeroAnswer                   ;acknowledge reception with single zero
;--------------------------
DoGetDataPortDirection:
                in      temp0,DDRB                      ;read direction of DDRB
                sts     AnswerArray,temp0               ;to array AnswerArray
                in      temp0,DDRC                      ;read direction of DDRC
                sts     AnswerArray+1,temp0             ;to array AnswerArray
                in      temp0,DDRD                      ;read direction of DDRD
                sts     AnswerArray+2,temp0             ;to array AnswerArray
                ldi     ZL,AnswerArray                  ;sending is value from AnswerArray
                ldi     temp0,0x81                      ;RAMREAD=1 - reading from RAM
                mov     RAMread,temp0                   ;(highest bit set to 1 - to zero RAMread immediatelly)
                ldi     temp0,3                         ;sending are three bytes
                rjmp    ComposeEndXXXDescriptor         ;and prepare data
;--------------------------
DoSetOutDataPort:
                lds     temp1,InputBufferBegin+7        ;fourth parameter - bit mask - which port(s) to change

                lds     temp0,InputBufferBegin+4        ;first parameter - value of data bits PORTB
                andi    temp0,0b00111100                ;mask unused pins
                sbrc    temp1,0                         ;if bit0 is zero - don't change port state
                out     PORTB,temp0                     ;and update data port

                lds     temp0,InputBufferBegin+5        ;second parameter - value of data bits PORTC
                sbrc    temp1,1                         ;if bit1 is zero - don't change port state
                out     PORTC,temp0                     ;and update data port

                lds     temp0,InputBufferBegin+6        ;third parameter - value of data bits PORTD
                andi    temp0,0b11111000                ;mask unused pins
                ori     temp0,0b00000011                ;mask unused pins
                sbrc    temp1,2                         ;if bit2 is zero - don't change port state
                out     PORTD,temp0                     ;and update data port

                rjmp    OneZeroAnswer                   ;acknowledge reception with single zero
;--------------------------
DoGetOutDataPort:
                in      temp0,PORTB                     ;read PORTB
                sts     AnswerArray,temp0               ;to array AnswerArray
                in      temp0,PORTC                     ;read PORTC
                sts     AnswerArray+1,temp0             ;to array AnswerArray
                in      temp0,PORTD                     ;read PORTD
                sts     AnswerArray+2,temp0             ;to array AnswerArray
                ldi     ZL,AnswerArray                  ;sending is value from AnswerArray
                ldi     temp0,0x81                      ;RAMREAD=1 - reading from RAM
                mov     RAMread,temp0                   ;(highest bit set to 1 - to zero RAMread immediatelly)
                ldi     temp0,3                         ;sending are three bytes
                rjmp    ComposeEndXXXDescriptor         ;and prepare data
;--------------------------
DoGetInDataPort:
                in      temp0,PINB                      ;read PINB
                sts     AnswerArray,temp0               ;to array AnswerArray
                in      temp0,PINC                      ;read PINC
                sts     AnswerArray+1,temp0             ;to array AnswerArray
                in      temp0,PIND                      ;read PIND
                sts     AnswerArray+2,temp0             ;to array AnswerArray
                ldi     ZL,AnswerArray                  ;sending is value from AnswerArray
                ldi     temp0,0x81                      ;RAMREAD=1 - reading from RAM
                mov     RAMread,temp0                   ;(highest bit set to 1 - to zero RAMread immediatelly)
                ldi     temp0,3                         ;sending are three bytes
                rjmp    ComposeEndXXXDescriptor         ;and prepare data
;----------------------------------------------------------------------------------------
 DoGetIn:
                ldi     ZL,0                            ;sending value in R0
                ldi     temp0,0x81                      ;RAMread=1 - reading from RAM
                mov     RAMread,temp0                   ;(highest bit set to 1 - to zero RAMread immediatelly)
                ldi     temp0,1                         ;send only single byte
                rjmp    ComposeEndXXXDescriptor         ;and prepare data
;----------------------------------------------------------------------------------------
DoEEPROMRead:
                lds     ZL,InputBufferBegin+4           ;first parameter - offset in EEPROM
                lds     ZH,InputBufferBegin+5
                ldi     temp0,2
                mov     RAMread,temp0                   ;RAMREAD=2 - reading from EEPROM
                ldi     temp0,E2END+1                   ;number my byte answers to temp0 - entire length of EEPROM
                rjmp    ComposeEndXXXDescriptor         ;otherwise prepare data
;--------------------------
DoEEPROMWrite:
                lds     ZL,InputBufferBegin+4           ;first parameter - offset in EEPROM (address)
                lds     ZH,InputBufferBegin+5
                lds     R0,InputBufferBegin+6           ;second parameter - data to store to EEPROM (data)
                out     EEAR,ZL                         ;set the address of EEPROM
                out     EEARH,ZH
                out     EEDR,R0                         ;set the data to EEPROM
                cli                                     ;disable interrupt
                sbi     EECR,EEMWE                      ;set the master write enable
                sei                                     ;enable interrupt (next instruction is performed)
                sbi     EECR,EEWE                       ;write
 WaitForEEPROMReady:
                sbic    EECR,EEWE                       ;wait to the end of write
                rjmp    WaitForEEPROMReady              ;in loop (max cca 4ms) (because of possible next reading/writing)
                rjmp    OneZeroAnswer                   ;acknowledge reception with single zero
;--------------------------
DoRS232Send:
                lds     temp0,InputBufferBegin+4        ;first parameter - value transmitted to RS232
                out     UDR,temp0                       ;transmit data to UART
 WaitForRS232Send:
                sbis    UCR,TXEN                        ;if disabled UART transmitter
                rjmp    OneZeroAnswer                   ;then finish - protection because loop lock in AT90S2323/2343
                sbis    USR,TXC                         ;wait for transmition finish
                rjmp    WaitForRS232Send
                rjmp    OneZeroAnswer                   ;acknowledge reception with single zero
;--------------------------
DoRS232Read:
                rjmp    TwoZeroAnswer                   ;only acknowledge reception with two zero
;--------------------------
DoSetRS232Baud:
                lds     temp0,InputBufferBegin+4        ;first parameter - value of baudrate of RS232
                lds     temp1,InputBufferBegin+6        ;second parameter - baudrate of RS232 - high byte
                cbr     temp1,1<<URSEL                  ;writing will be baudrate high byte (no UCSRC)
                out     UBRRH,temp1                     ;set the speed of UART high byte
                out     UBRR,temp0                      ;set the speed of UART low byte
                rjmp    OneZeroAnswer                   ;acknowledge reception with single zero
;--------------------------
DoGetRS232Baud:
                in      temp0,UBRR                      ;return speed of UART Lo
                sts     AnswerArray,temp0
                in      temp0,UBRRH                     ;return speed of UART Hi
                sts     AnswerArray+1,temp0             ;to array AnswerArray
                ldi     ZL,AnswerArray                  ;sending is value from AnswerArray
                ldi     temp0,0x81                      ;RAMREAD=1 - reading from RAM
                mov     RAMread,temp0                   ;(highest bit set to 1 - to zero RAMread immediatelly)
                ldi     temp0,2                         ;sending are two bytes
                rjmp    ComposeEndXXXDescriptor         ;and prepare data
;--------------------------
DoGetRS232Buffer:
                cbi     UCR,RXCIE                       ;disable interrupt from UART receiving
                nop
                lds     temp0,RS232LengthPosPtr
                lds     temp1,RS232LengthPosPtr+1       ;obtain buffer length of RS232 code
                sbi     UCR,RXCIE                       ;enable interrupt from UART receiving

                cpi     temp0,0                         ;if this isn't RS232 Buffer empty
                brne    SomeRS232Send                   ;then send it
                cpi     temp1,0                         ;if this isn't RS232 Buffer empty
                brne    SomeRS232Send                   ;then send it
                rjmp    OneZeroAnswer                   ;otherwise nothing send and acknowledge reception with single zero
 SomeRS232Send:
                lds     ACC,InputBufferBegin+8          ;number of requiring bytes to ACC
                ldi     temp2,2                         ;number of possible bytes (plus word of buffer length)
                add     temp0,temp2
                ldi     temp2,0
                adc     temp1,temp2
                cpi     temp1,0                         ;if is MSB>0
                brne    AsRequiredGetRS232Buffer        ;transmit as many as requested
                cp      ACC,temp0                       ;if no requested more that I can send
                brcc    NoShortGetRS232Buffer           ;transmit as many as requested
 AsRequiredGetRS232Buffer:
                mov     temp0,ACC
                ldi     temp1,0
 NoShortGetRS232Buffer:
                subi    temp0,2                         ;substract word length
                sbci    temp1,0
                lds     temp2,RS232ReadPosPtr           ;obtain index of reading of buffer of RS232 code
                lds     temp3,RS232ReadPosPtr+1
                add     temp2,temp0                     ;obtain where is end
                adc     temp3,temp1
                cpi     temp3,HIGH(RS232BufferEnd+1)    ;if it would overflow
                brlo    ReadNoOverflow                  ;
                brne    ReadOverflow                    ;if yes - skip to overflow

                cpi     temp2,LOW(RS232BufferEnd+1)     ;otherwise compare LSB
                brlo    ReadNoOverflow                  ;and do the same
 ReadOverflow:
                subi    temp2,LOW(RS232BufferEnd+1)     ;caculate how many not transfered
                sbci    temp3,HIGH(RS232BufferEnd+1)    ;caculate how many not transfered
                sub     temp0,temp2                     ;and with this short length of reading
                sbc     temp1,temp3                     ;and with this short length of reading
                ldi     temp2,LOW(RS232FIFOBegin)       ;and start from zero
                ldi     temp3,HIGH(RS232FIFOBegin)      ;and start from zero
 ReadNoOverflow:
                lds     ZL,RS232ReadPosPtr              ;obtain index of reading of buffer of RS232 code
                lds     ZH,RS232ReadPosPtr+1            ;obtain index of reading of buffer of RS232 code

                sts     RS232ReadPosPtr,temp2           ;write new index of reading of buffer of RS232 code
                sts     RS232ReadPosPtr+1,temp3         ;write new index of reading of buffer of RS232 code
                sbiw    ZL,2                            ;space for length data - transmitted as first word

                cbi     UCR,RXCIE                       ;disable interrupt from UART receiving
                inc     RAMread                         ;RAMread=1 reading from RAM
                lds     temp2,RS232LengthPosPtr
                lds     temp3,RS232LengthPosPtr+1       ;obtain buffer length of RS232 code
                sub     temp2,temp0                     ;decrement buffer length
                sbc     temp3,temp1
                sts     RS232LengthPosPtr,temp2         ;write new buffer length of RS232 code
                sts     RS232LengthPosPtr+1,temp3
                sbi     UCR,RXCIE                       ;enable interrupt from UART receiving

                st      Z+,temp2                        ;and save real length to packet
                st      Z,temp3                         ;and save real length to packet
                sbiw    ZL,1                            ;and set to begin
                inc     temp0                           ;and about this word increment number of transmited bytes (buffer length)
                inc     temp0
                rjmp    ComposeEndXXXDescriptor         ;and prepare data
;------------------------------------------------------------------------------------------
DoSetRS232DataBits:
                lds     temp0,InputBufferBegin+4        ;first parameter - data bits 0=5db, 1=6db, 2=7db, 3=8db
                cpi     temp0,DataBits8                 ;if to set 8-bits communication
                breq    Databits8or9Set                 ;then don't change 8/9 bit communication
                in      temp1,UCSRB                     ;otherwise load UCSRB
                cbr     temp1,(1<<UCSZ2)                ;clear 9-bit communication
                out     UCSRB,temp1                     ;and write back
 Databits8or9Set:
                rcall   RS232DataBitsLocal
                rjmp    OneZeroAnswer                   ;acknowledge reception with single zero
 RS232DataBitsLocal:
                rcall   GetUCSRCtotemp1
                bst     temp0,0                         ;set the UCSZ0
                bld     temp1,UCSZ0
                bst     temp0,1                         ;set the UCSZ1
                bld     temp1,UCSZ1
                rcall   Settemp1toUCSRC
                ret
 GetUCSRCtotemp1:
                cli                                     ;obtain UCSRC
                in      temp1,UBRRH
                in      temp1,UCSRC                     ;to temp1
                sei
                nop                                     ;for to enable possible interrupt waiting before ret instruction (ret has long duration)
                ret
 Settemp1toUCSRC:
                sbr     temp1,(1<<URSEL)                ;will be writing to UCSRC
                out     UCSRC,temp1                     ;and write back to register with new UCSZ0 and UCSZ1
                ret
;------------------------------------------------------------------------------------------
DoGetRS232DataBits:
                rcall   GetUCSRCtotemp1
                clr     temp0                           ;clear answer
                bst     temp1,UCSZ0                     ;obtain UCSZ0
                bld     temp0,0                         ;and save to bit 0
                bst     temp1,UCSZ1                     ;obtain UCSZ1
                bld     temp0,1                         ;and save to bit 1
                mov     R0,temp0                        ;return number of databits in R0
                rjmp    DoGetIn                         ;and finish
;------------------------------------------------------------------------------------------
DoSetRS232Parity:
                lds     temp0,InputBufferBegin+4        ;first parameter - parity: 0=none, 1=odd, 2=even, 3=mark, 4=space
                cpi     temp0,3
                brcc    StableParity
                rcall   GetUCSRCtotemp1
                cbr     temp1,(1<<UPM1)|(1<<UPM0)       ;clear parity bits
                cpi     temp0,ParityNone                ;if none
                breq    SetParityOut
                sbr     temp1,(1<<UPM1)
                cpi     temp0,ParityEven                ;if even
                breq    SetParityOut
                sbr     temp1,(1<<UPM0)
                cpi     temp0,ParityOdd                 ;if odd
                brne    ParityErrorAnswer
 SetParityOut:
                rcall   Settemp1toUCSRC
                in      temp1,UCSRB                     ;load UCSRB
                cbr     temp1,(1<<UCSZ2)                ;if is 9-bits communication then change it under 9 bits
                out     UCSRB,temp1                     ;and write back
                rjmp    OneZeroAnswer                   ;acknowledge reception with single zero
 StableParity:
                in      temp1,UCSRB                     ;change transmiting parity bit TXB8
                bst     temp0,0                         ;load lowest bit
                bld     temp1,TXB8                      ;and save to its place TXB8
                sbr     temp1,(1<<UCSZ2)                ;set the UCSZ2 bit - 9 bits communication
                out     UCSRB,temp1                     ;changed TXB8 and UCSZ2 write to UCSRB

                ldi     temp0,3                         ;set the 9-databit
                rcall   RS232DataBitsLocal              ;and return in temp1 contents UCSRC
                cbr     temp1,(1<<UPM1)|(1<<UPM0)       ;disable parity
                rcall   Settemp1toUCSRC
                rjmp    OneZeroAnswer                   ;acknowledge reception with single zero
 ParityErrorAnswer:
                rjmp    TwoZeroAnswer                   ;acknowledge reception with two zero
;------------------------------------------------------------------------------------------
DoGetRS232Parity:
                in      temp1,UCSRB                     ;load UCSRB
                sbrc    temp1,UCSZ2                     ;if is 9-bits communication
                rjmp    ParityIsStable                  ;then parity is space or mark

                rcall   GetUCSRCtotemp1
                cbr     temp1,~((1<<UPM0)|(1<<UPM1))    ;and let nonzero only parity bits

                cpi     temp1,(1<<UPM0)|(1<<UPM1)       ;if are both set
                ldi     temp0,ParityOdd                 ;this is odd parity
                breq    RetGetParity                    ;and finish
                cpi     temp1,(1<<UPM1)                 ;if is UPM1 set
                ldi     temp0,ParityEven                ;this is even parity
                breq    RetGetParity                    ;and finish
                ldi     temp0,ParityNone                ;otherwise is that none parity
                rjmp    RetGetParity                    ;and finish
 ParityIsStable:
                bst     temp1,TXB8                      ;obtain what is 9-th bit
                ldi     temp0,ParityMark                ;prepare mark answer
                brts    RetGetParity                    ;if is 1 then return mark
                ldi     temp0,ParitySpace               ;otherwise return space
 RetGetParity:
                mov     R0,temp0                        ;answer move from temp0 to R0
                rjmp    DoGetIn                         ;and finish
;------------------------------------------------------------------------------------------
DoSetRS232StopBits:
                lds     temp0,InputBufferBegin+4        ;first parameter - stop bit 0=1stopbit 1=2stopbits
                rcall   GetUCSRCtotemp1
                bst     temp0,0                         ;and lowest bit from parameter
                bld     temp1,USBS                      ;save as stopbit
                rcall   Settemp1toUCSRC
                rjmp    OneZeroAnswer                   ;acknowledge reception with single zero
;------------------------------------------------------------------------------------------
DoGetRS232StopBits:
                rcall   GetUCSRCtotemp1
                clr     R0                              ;clear answer
                bst     temp1,USBS                      ;and bit USBS
                bld     R0,0                            ;write to answer
                rjmp    DoGetIn                         ;and finish
;------------------------------------------------------------------------------------------
;------------------------------ END USER FUNCTIONS ------------------------------------- END USER FUNCTIONS ------------------------------

OneZeroAnswer:                  ;send single zero
                ldi     temp0,1                         ;number of my bytes answers to temp0
                rjmp    ComposeGET_STATUS2
;------------------------------ STANDARD USB REQUESTS ---------------------------------- STANDARD USB REQUESTS ------------------------------
StandardRequest:
                cpi     temp1,GET_STATUS                ;
                breq    ComposeGET_STATUS               ;

                cpi     temp1,CLEAR_FEATURE             ;
                breq    ComposeCLEAR_FEATURE            ;

                cpi     temp1,SET_FEATURE               ;
                breq    ComposeSET_FEATURE              ;

                cpi     temp1,SET_ADDRESS               ;if to set address
                breq    ComposeSET_ADDRESS              ;set the address

                cpi     temp1,GET_DESCRIPTOR            ;if requested descriptor
                breq    ComposeGET_DESCRIPTOR           ;generate it

                cpi     temp1,SET_DESCRIPTOR            ;
                breq    ComposeSET_DESCRIPTOR           ;

                cpi     temp1,GET_CONFIGURATION         ;
                breq    ComposeGET_CONFIGURATION        ;

                cpi     temp1,SET_CONFIGURATION         ;
                breq    ComposeSET_CONFIGURATION        ;

                cpi     temp1,GET_INTERFACE             ;
                breq    ComposeGET_INTERFACE            ;

                cpi     temp1,SET_INTERFACE             ;
                breq    ComposeSET_INTERFACE            ;

                cpi     temp1,SYNCH_FRAME               ;
                breq    ComposeSYNCH_FRAME              ;
                                                        ;if not found known request
                rjmp    ZeroDATA1Answer                 ;if that was something unknown, then prepare zero answer

ComposeSET_ADDRESS:
                lds     temp1,InputBufferBegin+4        ;new address to temp1
                rcall   SetMyNewUSBAddresses            ;and compute NRZI and bitstuffing coded adresses
                ldi     State,AddressChangeState        ;set state for Address changing
                rjmp    ZeroDATA1Answer                 ;send zero answer

ComposeSET_CONFIGURATION:
                lds     temp0,InputBufferBegin+4        ;number of configuration to variable ConfigByte
                sts     ConfigByte,temp0                ;
ComposeCLEAR_FEATURE:
ComposeSET_FEATURE:
ComposeSET_INTERFACE:
ZeroStringAnswer:
                rjmp    ZeroDATA1Answer                 ;send zero answer
ComposeGET_STATUS:
TwoZeroAnswer:
                ldi     temp0,2                         ;number of my bytes answers to temp0
ComposeGET_STATUS2:
                ldi     ZH, high(StatusAnswer<<1)       ;ROMpointer to  answer
                ldi     ZL,  low(StatusAnswer<<1)
                rjmp    ComposeEndXXXDescriptor         ;and complete
ComposeGET_CONFIGURATION:
                lds     temp0,ConfigByte
                and     temp0,temp0                     ;if I am unconfigured
                breq    OneZeroAnswer                   ;then send single zero - otherwise send my configuration
                ldi     temp0,1                         ;number of my bytes answers to temp0
                ldi     ZH, high(ConfigAnswerMinus1<<1) ;ROMpointer to  answer
                ldi     ZL,  low(ConfigAnswerMinus1<<1)+1
                rjmp    ComposeEndXXXDescriptor         ;and complete
ComposeGET_INTERFACE:
                ldi     ZH, high(InterfaceAnswer<<1)    ;ROMpointer to answer
                ldi     ZL,  low(InterfaceAnswer<<1)
                ldi     temp0,1                         ;number of my bytes answers to temp0
                rjmp    ComposeEndXXXDescriptor         ;and complete
ComposeSYNCH_FRAME:
ComposeSET_DESCRIPTOR:
                rcall   ComposeSTALL
                ret
ComposeGET_DESCRIPTOR:
                lds     temp1,InputBufferBegin+5        ;DescriptorType to temp1
                cpi     temp1,DEVICE                    ;DeviceDescriptor
                breq    ComposeDeviceDescriptor         ;
                cpi     temp1,CONFIGURATION             ;ConfigurationDescriptor
                breq    ComposeConfigDescriptor         ;
                cpi     temp1,STRING                    ;StringDeviceDescriptor
                breq    ComposeStringDescriptor         ;
                ret
ComposeDeviceDescriptor:
                ldi     ZH, high(DeviceDescriptor<<1)   ;ROMpointer to descriptor
                ldi     ZL,  low(DeviceDescriptor<<1)
                ldi     temp0,0x12                      ;number of my bytes answers to temp0
                rjmp    ComposeEndXXXDescriptor         ;and complete
ComposeConfigDescriptor:
                ldi     ZH, high(ConfigDescriptor<<1)   ;ROMpointer to descriptor
                ldi     ZL,  low(ConfigDescriptor<<1)
                ldi     temp0,9+9+7                     ;number of my bytes answers to temp0
ComposeEndXXXDescriptor:
                lds     TotalBytesToSend,InputBufferBegin+8     ;number of requested bytes to TotalBytesToSend
                cp      TotalBytesToSend,temp0                  ;if not requested more than I can send
                brcs    HostConfigLength                ;transmit the requested number
                mov     TotalBytesToSend,temp0          ;otherwise send number of my answers
HostConfigLength:
                mov     temp0,TotalBytesToSend          ;
                clr     TransmitPart                    ;zero the number of 8 bytes answers
                andi    temp0,0b00000111                ;if is length divisible by 8
                breq    Length8Multiply                 ;then not count one answer (under 8 byte)
                inc     TransmitPart                    ;otherwise count it
Length8Multiply:
                mov     temp0,TotalBytesToSend          ;
                lsr     temp0                           ;length of 8 bytes answers will reach
                lsr     temp0                           ;integer division by 8
                lsr     temp0
                add     TransmitPart,temp0              ;and by addition to last non entire 8-bytes to variable TransmitPart
                ldi     temp0,DATA0PID                  ;DATA0 PID - in the next will be toggled to DATA1PID in load descriptor
                sts     OutputBufferBegin+1,temp0       ;store to output buffer
                rjmp    ComposeNextAnswerPart
ComposeStringDescriptor:
                ldi     temp1,4+8                       ;if RAMread=4(insert zeros from ROM reading) + 8(behind first byte no load zero)
                mov     RAMread,temp1
                lds     temp1,InputBufferBegin+4        ;DescriptorIndex to temp1
                cpi     temp1,0                         ;LANGID String
                breq    ComposeLangIDString             ;
                cpi     temp1,2                         ;DevNameString
                breq    ComposeDevNameString            ;
                brcc    ZeroStringAnswer                ;if is DescriptorIndex higher than 2 - send zero answer
                                                        ;otherwise is VendorString
ComposeVendorString:
                ldi     ZH, high(VendorStringDescriptor<<1)     ;ROMpointer to descriptor
                ldi     ZL,  low(VendorStringDescriptor<<1)
                ldi     temp0,(VendorStringDescriptorEnd-VendorStringDescriptor)*4-2    ;number of my bytes answers to temp0
                rjmp    ComposeEndXXXDescriptor         ;and complete
ComposeDevNameString:
                ldi     ZH, high(DevNameStringDescriptor<<1)    ;ROMpointer to descriptor
                ldi     ZL,  low(DevNameStringDescriptor<<1)
                ldi     temp0,(DevNameStringDescriptorEnd-DevNameStringDescriptor)*4-2  ;number of my bytes answers to temp0
                rjmp    ComposeEndXXXDescriptor         ;and complete
ComposeLangIDString:
                clr     RAMread
                ldi     ZH, high(LangIDStringDescriptor<<1)     ;ROMpointer to descriptor
                ldi     ZL,  low(LangIDStringDescriptor<<1)
                ldi     temp0,(LangIDStringDescriptorEnd-LangIDStringDescriptor)*2;number of my bytes answers to temp0
                rjmp    ComposeEndXXXDescriptor         ;and complete
;------------------------------------------------------------------------------------------
ZeroDATA1Answer:
                rcall   ComposeZeroDATA1PIDAnswer
                ret
;------------------------------------------------------------------------------------------
SetMyNewUSBAddresses:           ;set new USB addresses in NRZI coded
                mov     temp2,temp1             ;address to temp2 and temp1 and temp3
                mov     temp3,temp1             ;
                cpi     temp1,0b01111111        ;if address contains less than 6 ones
                brne    NewAddressNo6ones       ;then don't add bitstuffing
                ldi     temp1,0b10111111        ;else insert one zero - bitstuffing
 NewAddressNo6ones:
                andi    temp3,0b00000111        ;mask 3 low bits of Address
                cpi     temp3,0b00000111        ;and if 3 low bits of Address is no all ones
                brne    NewAddressNo3ones       ;then no change address
                                                ;else insert zero after 3-rd bit (bitstuffing)
                sec                             ;carry
                rol     temp2                   ;rotate left
                andi    temp2,0b11110111        ;and inserted zero after 3-rd bit
 NewAddressNo3ones:
                sts     MyOutAddressSRAM,temp2  ;store new non-coded address Out (temp2)
                                                ;and now perform NRZI coding
                rcall   NRZIforAddress          ;NRZI for AddressIn (in temp1)
                sts     MyInAddressSRAM,ACC     ;store NRZI coded AddressIn

                lds     temp1,MyOutAddressSRAM  ;load non-coded address Out (in temp1)
                rcall   NRZIforAddress          ;NRZI for AddressOut
                sts     MyOutAddressSRAM,ACC    ;store NRZI coded AddressOut

                ret                             ;and return
;------------------------------------------------------------------------------------------
NRZIforAddress:
                clr     ACC                     ;original answer state - of my nNRZI USB address
                ldi     temp2,0b00000001        ;mask for xoring
                ldi     temp3,8                 ;bits counter
SetMyNewUSBAddressesLoop:
                mov     temp0,ACC               ;remember final answer
                ror     temp1                   ;to carry transmitting bit LSB (in direction firstly LSB then MSB)
                brcs    NoXORBits               ;if one - don't change state
                eor     temp0,temp2             ;otherwise state will be changed according to last bit of answer
NoXORBits:
                ror     temp0                   ;last bit of changed answer to carry
                rol     ACC                     ;and from carry to final answer to the LSB place (and reverse LSB and MSB order)
                dec     temp3                   ;decrement bits counter
                brne    SetMyNewUSBAddressesLoop ;if bits counter isn't zero repeat transmitting with next bit
                ret
;------------------------------------------------------------------------------------------
;-------------------------- END DATA ENCRYPTION USB REQUESTS -----------------------------

PrepareOutContinuousBuffer:
                rcall   PrepareContinuousBuffer
                rcall   MakeOutBitStuff
                ret
;------------------------------------------------------------------------------------------
PrepareContinuousBuffer:
                mov     temp0,TransmitPart
                cpi     temp0,1
                brne    NextAnswerInBuffer              ;if buffer empty
                rcall   ComposeZeroAnswer               ;prepare zero answer
                ret
NextAnswerInBuffer:
                dec     TransmitPart                    ;decrement general length of answer
ComposeNextAnswerPart:
                mov     temp1,TotalBytesToSend  ;decrement number of bytes to transmit
                subi    temp1,8                 ;is is necessary to send more as 8 byte
                ldi     temp3,8                 ;if yes - send only 8 byte
                brcc    Nad8Bytov
                mov     temp3,TotalBytesToSend  ;otherwise send only given number of bytes
                clr     TransmitPart
                inc     TransmitPart            ;and this will be last answer
Nad8Bytov:
                mov     TotalBytesToSend,temp1  ;decremented number of bytes to TotalBytesToSend
                rcall   LoadXXXDescriptor
                ldi     ByteCount,2             ;length of output buffer (only SOP and PID)
                add     ByteCount,temp3         ;+ number of bytes
                rcall   AddCRCOut               ;addition of CRC to buffer
                inc     ByteCount               ;length of output buffer + CRC16
                inc     ByteCount
                ret                             ;finish
;------------------------------------------------------------------------------------------
.equ    USBversion              =0x0101         ;for what version USB is that (1.01)
.equ    VendorUSBID             =0x03EB         ; vendor identifier (Atmel=0x03EB)
.equ    DeviceUSBID             =0x21FF         ;product identifier (USB to RS232 converter ATmega8=0x21FF)
.equ    DeviceVersion           =0x0003         ;version number of product (version=0.03)
                                                ;(0.01=AT90S2313 Infra buffer)
                                                ;(0.02=AT90S2313 RS232 buffer 32bytes)
                                                ;(0.03=ATmega8 RS232 buffer 800bytes)
.equ    MaxUSBCurrent           =50             ;current consumption from USB (50mA) - together with MAX232
;------------------------------------------------------------------------------------------
DeviceDescriptor:
                .db     0x12,0x01               ;0 byte - size of descriptor in byte
                                                ;1 byte - descriptor type: Device descriptor
                .dw     USBversion              ;2,3 byte - version USB LSB (1.00)
                .db     0x00,0x00               ;4 byte - device class
                                                ;5 byte - subclass
                .db     0x00,0x08               ;6 byte - protocol code
                                                ;7 byte - FIFO size in bytes
                .dw     VendorUSBID             ;8,9 byte - vendor identifier (Cypress=0x04B4)
                .dw     DeviceUSBID             ;10,11 byte - product identifier (teplomer=0x0002)
                .dw     DeviceVersion           ;12,13 byte - product version number (verzia=0.01)
                .db     0x01,0x02               ;14 byte - index of string "vendor"
                                                ;15 byte - index of string "product"
                .db     0x00,0x01               ;16 byte - index of string "serial number"
                                                ;17 byte - number of possible configurations
DeviceDescriptorEnd:
;------------------------------------------------------------------------------------------
ConfigDescriptor:
                .db     0x9,0x02                ;length, descriptor type
ConfigDescriptorLength:
                .dw     9+9+7                   ;entire length of all descriptors
        ConfigAnswerMinus1:                     ;for sending the number - congiguration number (attention - addition of 1 required)
                .db     1,1                     ;numInterfaces, congiguration number
                .db     0,0x80                  ;string index, attributes; bus powered
                .db     MaxUSBCurrent/2,0x09    ;current consumption, interface descriptor length
                .db     0x04,0                  ;interface descriptor; number of interface
        InterfaceAnswer:                        ;for sending number of alternatively interface
                .db     0,1                     ;alternatively interface; number of endpoints except EP0
        StatusAnswer:                           ;2 zero answers (saving ROM place)
                .db     0,0                     ;interface class; interface subclass
                .db     0,0                     ;protocol code; string index
                .db     0x07,0x5                ;length, descriptor type - endpoint
                .db     0x81,0                  ;endpoint address; transfer type
                .dw     0x08                    ;max packet size
                .db     10,0                    ;polling interval [ms]; dummy byte (for filling)
ConfigDescriptorEnd:
;------------------------------------------------------------------------------------------
LangIDStringDescriptor:
                .db     (LangIDStringDescriptorEnd-LangIDStringDescriptor)*2,3  ;length, type: string descriptor
                .dw     0x0409                  ;English
LangIDStringDescriptorEnd:
;------------------------------------------------------------------------------------------
VendorStringDescriptor:
                .db     (VendorStringDescriptorEnd-VendorStringDescriptor)*4-2,3        ;length, type: string descriptor
CopyRight:
                .db     "Ing. Igor Cesko http://www.cesko.host.sk"
CopyRightEnd:
VendorStringDescriptorEnd:
;------------------------------------------------------------------------------------------
DevNameStringDescriptor:
                .db     (DevNameStringDescriptorEnd-DevNameStringDescriptor)*4-2,3;length, type: string descriptor
                .db     "AVR309: USB to UART protocol converter"
DevNameStringDescriptorEnd:
;------------------------------------------------------------------------------------------
;********************************************************************
;* End of program
;********************************************************************
;------------------------------------------------------------------------------------------
;------------------------------------------------------------------------------------------
;********************************************************************
```

```
;* End of file
;*********************************************************************
```