

```

1 ;*****
2 ;* U S B   S T A C K   F O R   T H E   A V R   F A M I L Y
3 ;*
4 ;* File Name           : "USBtoRS232.asm"
5 ;* Title              : AVR309:USB to UART protocol converter
6 ;* Date               : 01.02.2004
7 ;* Version            : 2.8
8 ;* Target MCU         : ATmega8
9 ;* AUTHOR             : Ing. Igor Cesko
10 ;*                   : Slovakia
11 ;*                   : cesko@internet.sk
12 ;*                   : http://www.cesko.host.sk
13 ;*
14 ;* DESCRIPTION:
15 ;*   USB protocol implementation into MCU with noUSB interface:
16 ;*   Device:
17 ;*   Universal USB interface (3x8-bit I/O port + RS232 serial line + EEPROM)
18 ;*   + added RS232 FIFO buffer
19 ;*
20 ;* The timing is adapted for 12 MHz crystal
21 ;*
22 ;*
23 ;* to add your own functions - see section: TEMPLATE OF YOUR FUNCTION
24 ;*
25 ;* to customize device to your company you must change VendorUSB ID (VID)
26 ;* to VID assigned to your company (for more information see www.usb.org)
27 ;*
28 ;*****
29 .include "m8def.inc"
30 ;comment for AT90S2313
31 .equ    UCR                =UCSRB
32 .equ    UBRR               =UBRR1
33 .equ    EEAR               =EEAR1
34 .equ    USR                =UCSRA
35 .equ    E2END              =127
36 .equ    RAMEND128          =96+127
37
38 .equ    inputport          =PINB
39 .equ    outputport         =PORTB
40 .equ    USBdirection       =DDRB
41 .equ    DATApplus          =1           ;signal D+ on PB1
42 .equ    DATAmminus         =0           ;signal D- on PB0 - give on this pin pull-up 1.5kOhm
43 .equ    USBpinmask         =0b1111100 ;mask low 2 bit (D+,D-) on PB
44 .equ    USBpinmaskDplus    = ~(1<<DATApplus) ;mask D+ bit on PB
45 .equ    USBpinmaskDminus   = ~(1<<DATAmminus) ;mask D- bit on PB
46
47 .equ    TSOPPort           =PINB
48 .equ    TSOPpullupPort     =PORTB
49 .equ    TSOPPin            =2           ;signal OUT from IR sensor TSOP1738 on PB2
50
51 ;connecting LED diode LSB
52 ;connecting LED diode LSB (input)
53 ;input/output LED LSB
54 ;connecting LED diode MSB
55 ;connecting LED diode MSB (input)
56 ;input/output LED MSB
57 ;LED0 on pin PD3
58 ;LED1 on pin PD5

```

```

59 ;LED2 on pin PD6
60 ;LED3 on pin PB3
61 ;LED4 on pin PB4
62 ;LED5 on pin PB5
63 ;LED6 on pin PB6
64 ;LED7 on pin PB7
65
66 .equ      SOPbyte      =0b10000000      ;Start of Packet byte
67 .equ      DATA0PID    =0b11000011      ;PID for DATA0 field
68 .equ      DATA1PID    =0b01001011      ;PID for DATA1 field
69 .equ      OUTPID       =0b11100001      ;PID for OUT field
70 .equ      INPID        =0b01101001      ;PID for IN field
71 .equ      SOFPID       =0b10100101      ;PID for SOF field
72 .equ      SETUPPID     =0b00101101      ;PID for SETUP field
73 .equ      ACKPID       =0b11010010      ;PID for ACK field
74 .equ      NAKPID       =0b01011010      ;PID for NAK field
75 .equ      STALLPID     =0b00011110      ;PID for STALL field
76 .equ      PREPID       =0b00111100      ;PID for FOR field
77
78 .equ      nSOPbyte      =0b00000001      ;Start of Packet byte - reverse order
79 .equ      nDATA0PID     =0b11000011      ;PID for DATA0 field - reverse order
80 .equ      nDATA1PID     =0b11010010      ;PID for DATA1 field - reverse order
81 .equ      nOUTPID       =0b10000111      ;PID for OUT field - reverse order
82 .equ      nINPID        =0b10010110      ;PID for IN field - reverse order
83 .equ      nSOFPID       =0b10100101      ;PID for SOF field - reverse order
84 .equ      nSETUPPID     =0b10110100      ;PID for SETUP field - reverse order
85 .equ      nACKPID       =0b01001011      ;PID for ACK field - reverse order
86 .equ      nNAKPID       =0b01011010      ;PID for NAK field - reverse order
87 .equ      nSTALLPID     =0b01111000      ;PID for STALL field - reverse order
88 .equ      nPREPID       =0b00111100      ;PID for FOR field - reverse order
89
90 .equ      nNRZITokenPID =~0b10000000      ;PID mask for Token packet (IN,OUT,SOF,SETUP) - reverse order NRZI
91 .equ      nNRZISOPbyte  =~0b10101011      ;Start of Packet byte - reverse order NRZI
92 .equ      nNRZIDATA0PID =~0b11010111      ;PID for DATA0 field - reverse order NRZI
93 .equ      nNRZIDATA1PID =~0b11001001      ;PID for DATA1 field - reverse order NRZI
94 .equ      nNRZIOUTPID   =~0b10101111      ;PID for OUT field - reverse order NRZI
95 .equ      nNRZIINPID    =~0b10110001      ;PID for IN field - reverse order NRZI
96 .equ      nNRZISOFPID   =~0b10010011      ;PID for SOF field - reverse order NRZI
97 .equ      nNRZISETUPPID =~0b10001101      ;PID for SETUP field - reverse order NRZI
98 .equ      nNRZIACKPID   =~0b00100111      ;PID for ACK field - reverse order NRZI
99 .equ      nNRZINAKPID   =~0b00111001      ;PID for NAK field - reverse order NRZI
100 .equ      nNRZISTALLPID =~0b00000111      ;PID for STALL field - reverse order NRZI
101 .equ      nNRZIPREPID   =~0b01111101      ;PID for FOR field - reverse order NRZI
102 .equ      nNRZIADDR0    =~0b01010101      ;Address = 0 - reverse order NRZI
103
104                                     ;status bytes - State
105 .equ      BaseState      =0                ;
106 .equ      SetupState     =1                ;
107 .equ      InState        =2                ;
108 .equ      OutState       =3                ;
109 .equ      SOFState       =4                ;
110 .equ      DataState      =5                ;
111 .equ      AddressChangeState =6            ;
112
113                                     ;Flags of action
114 .equ      DoNone          =0
115 .equ      DoReceiveOutData =1
116 .equ      DoReceiveSetupData =2

```

```

117 .equ      DoPrepareOutContinuousBuffer      =3
118 .equ      DoReadySendAnswer                =4
119
120
121 .equ      CRC5poly                          =0b00101          ;CRC5 polynomial
122 .equ      CRC5zvysok                       =0b01100          ;CRC5 remainder after successful CRC5
123 .equ      CRC16poly                       =0b1000000000000101    ;CRC16 polynomial
124 .equ      CRC16zvysok                     =0b1000000000001101    ;CRC16 remainder after successful CRC16
125
126 .equ      MAXUSBBYTES                      =14                ;maximum bytes in USB input message
127 .equ      NumberOfFirstBits                =10                ;how many first bits allowed be longer
128 .equ      NoFirstBitsTimerOffset           =256-12800*12/1024    ;Timeout 12.8ms (12800us) to terminate after firsts bits
129 .equ      InitBaudRate                     =12000000/16/57600-1    ;UART on 57600 (for 12MHz=12000000Hz)
130
131 .equ      InputBufferBegin                  =RAMEND128-127      ;compare of receiving shift buffer
132 .equ      InputShiftBufferBegin            =InputBufferBegin+MAXUSBBYTES ;compare of receiving buffera
133
134 .equ      MyInAddressSRAM                   =InputShiftBufferBegin+MAXUSBBYTES
135 .equ      MyOutAddressSRAM                  =MyInAddressSRAM+1
136
137 .equ      OutputBufferBegin                 =RAMEND128-MAXUSBBYTES-2 ;compare of transmitting buffer
138 .equ      AckBufferBegin                   =OutputBufferBegin-3    ;compare of transmitting buffer Ack
139 .equ      NakBufferBegin                   =AckBufferBegin-3      ;compare of transmitting buffer Nak
140 .equ      ConfigByte                       =NakBufferBegin-1      ;0=unconfigured state
141 .equ      AnswerArray                      =ConfigByte-8          ;8 byte answer array
142 .equ      StackBegin                       =AnswerArray-1         ;low reservoir (stack is big cca 68 byte)
143
144 .equ      MAXRS232LENGTH                    =RAMEND-RAMEND128-10    ;maximum length RS232 code
145 .equ      RS232BufferBegin                  =RAMEND128+1          ;compare of buffer for RS232 - receiving
146 .equ      RS232BufferEnd                    =RS232BufferBegin+MAXRS232LENGTH
147 .equ      RS232ReadPosPtr                   =RS232BufferBegin+0
148 .equ      RS232WritePosPtr                  =RS232BufferBegin+2
149 .equ      RS232LengthPosPtr                 =RS232BufferBegin+4
150 .equ      RS232Reserved                     =RS232BufferBegin+6
151 .equ      RS232FIFOBegin                    =RS232BufferBegin+8
152
153
154
155 .def       RS232BufferFull                   =R1                ;flag of full RS232 buffer
156 .def       backupbitcount                   =R2                ;backup bitcount register in INT0 disconnected
157 .def       RAMread                          =R3                ;if reading from SRAM
158 .def       backupSREGTimer                  =R4                ;backup Flag register in Timer interrupt
159 .def       backupSREG                       =R5                ;backup Flag register in INT0 interrupt
160 .def       ACC                              =R6                ;accumulator
161 .def       lastBitstufNumber                 =R7                ;position in bitstuffing
162 .def       OutBitStuffNumber                 =R8                ;how many bits to send last byte - bitstuffing
163 .def       BitStuffInOut                     =R9                ;if insertion or deleting of bitstuffing
164 .def       TotalBytesToSend                  =R10               ;how many bytes to send
165 .def       TransmitPart                      =R11               ;order number of transmitting part
166 .def       InputBufferLength                 =R12               ;length prepared in input USB buffer
167 .def       OutputBufferLength                =R13               ;length answers prepared in USB buffer
168 .def       MyOutAddress                      =R14               ;my USB address (Out Packet) for update
169 .def       MyInAddress                       =R15               ;my USB address (In/SetupPacket)
170
171
172 .def       ActionFlag                        =R16                ;what to do in main program loop
173 .def       temp3                             =R17                ;temporary register
174 .def       temp2                             =R18                ;temporary register

```

```

175 .def      temp1          =R19          ;temporary register
176 .def      temp0          =R20          ;temporary register
177 .def      bitcount        =R21          ;counter of bits in byte
178 .def      ByteCount       =R22          ;counter of maximum number of received bytes
179 .def      inputbuf        =R23          ;receiver register
180 .def      shiftbuf        =R24          ;shift receiving register
181 .def      State           =R25          ;state byte of status of state machine
182 .def      RS232BufptrX    =R26          ;XL register - pointer to buffer of received IR codes
183 .def      RS232BufptrXH   =R27
184 .def      USBBufptrY      =R28          ;YL register - pointer to USB buffer input/output
185 .def      ROMBufptrZ      =R30          ;ZL register - pointer to buffer of ROM data
186
187
188 ;requirements on descriptors
189 .equ      GET_STATUS       =0
190 .equ      CLEAR_FEATURE    =1
191 .equ      SET_FEATURE      =3
192 .equ      SET_ADDRESS      =5
193 .equ      GET_DESCRIPTOR    =6
194 .equ      SET_DESCRIPTOR    =7
195 .equ      GET_CONFIGURATION =8
196 .equ      SET_CONFIGURATION =9
197 .equ      GET_INTERFACE     =10
198 .equ      SET_INTERFACE     =11
199 .equ      SYNCH_FRAME      =12
200
201 ;descriptor types
202 .equ      DEVICE           =1
203 .equ      CONFIGURATION    =2
204 .equ      STRING           =3
205 .equ      INTERFACE        =4
206 .equ      ENDPOINT         =5
207
208 ;databits
209 .equ      DataBits5        =0
210 .equ      DataBits6        =1
211 .equ      DataBits7        =2
212 .equ      DataBits8        =3
213
214 ;parity
215 .equ      ParityNone       =0
216 .equ      ParityOdd        =1
217 .equ      ParityEven       =2
218 .equ      ParityMark       =3
219 .equ      ParitySpace      =4
220
221 ;stopbits
222 .equ      StopBit1         =0
223 .equ      StopBit2         =1
224
225 ;user function start number
226 .equ      USER_FNC_NUMBER =100
227
228
229 ;-----
230 ;*****
231 ;* Interrupt table
232 ;*****

```

```

233 .cseg
234 ;-----
235 .org 0                                ;after reset
236     rjmp    reset
237 ;-----
238 .org INT0addr                        ;external interrupt INT0
239     rjmp    INT0handler
240 ;-----
241 .org URXCaddr                        ;receiving from serial line
242     push    temp0
243     cbi     UCR,RXCIE                ;disable interrupt from UART receiving
244     sei     ;enable interrupts to service USB
245     in      temp0,UDR                ;put to temp0 received data from UART
246     in      backupSREGTimer,SREG    ;backup SREG
247     push    temp2
248     push    temp3
249     lds     temp2,RS232LengthPosPtr
250     lds     temp3,RS232LengthPosPtr+1 ;determine length of RS232 code buffer
251     cpi     temp3,HIGH(RS232BufferEnd-RS232FIFOBegin-1) ;if the buffer would overflow
252     brlo    FIFOBufferNoOverflow    ;if not overflow then write to FIFO
253     brne    NoIncRS232BufferFull    ;if buffer would overflow, then prevent of overwriting
254     ;otherwise (if equal) still compare Lo bytes
255     cpi     temp2,LOW(RS232BufferEnd-RS232FIFOBegin-1) ;if buffer would overflow (Lo byte)
256     brcc    NoIncRS232BufferFull    ;then prevent of overwriting
257 FIFOBufferNoOverflow:
258     push    RS232BufptrX
259     push    RS232BufptrXH
260     lds     RS232BufptrX,RS232WritePosPtr ;set position to begin of buffer write RS232 code
261     lds     RS232BufptrXH,RS232WritePosPtr+1 ;set position to begin of buffer write RS232 code
262
263     st      X+,temp0                  ;and save it to buffer
264     cpi     RS232BufptrXH,HIGH(RS232BufferEnd+1) ;if not reached maximum of RS232 buffer
265     brlo    NoUARTBufferOverflow    ;then continue
266     brne    UARTBufferOverflow      ;check althen LSB
267     cpi     RS232BufptrX,LOW(RS232BufferEnd+1) ;if not reached maximum of RS232 buffer
268     brlo    NoUARTBufferOverflow    ;then continue
269 UARTBufferOverflow:
270     ldi     RS232BufptrX,LOW(RS232FIFOBegin) ;otherwise set position to buffer begin
271     ldi     RS232BufptrXH,HIGH(RS232FIFOBegin) ;otherwise set position to buffer begin
272 NoUARTBufferOverflow:
273     sts     RS232WritePosPtr,RS232BufptrX ;save new offset of buffer write RS232 code
274     sts     RS232WritePosPtr+1,RS232BufptrXH ;save new offset of buffer write RS232 code
275     ldi     temp0,1                   ;increment length of RS232 buffer
276     add     temp2,temp0
277     ldi     temp0,0
278     adc     temp3,temp0
279     sts     RS232LengthPosPtr,temp2    ;save length of buffer RS232 code
280     sts     RS232LengthPosPtr+1,temp3 ;save length of buffer RS232 code
281     pop     RS232BufptrXH
282     pop     RS232BufptrX
283 NoIncRS232BufferFull:
284     pop     temp3
285     pop     temp2
286     pop     temp0
287     out     SREG,backupSREGTimer      ;restore SREG
288     cli     ;disable interrupt because to prevent reentrant interrupt call
289     sbi     UCR,RXCIE                ;enable interrupt from receiving of UART
290     reti

```

```

291 ;-----
292 ;*****
293 ;* Init program
294 ;*****
295 ;-----
296 reset:                ;initialization of processor and variables to right values
297     ldi    temp0,StackBegin    ;initialization of stack
298     out    SPL,temp0
299
300     clr    XH                  ;RS232 pointer
301     clr    YH                  ;USB pointer
302     clr    ZH                  ;ROM pointer
303     ldi    temp0,LOW(RS232FIFOBegin) ;set Low to begin of buffer
304     sts    RS232ReadPosPtr,temp0 ;zero index of reading
305     sts    RS232WritePosPtr,temp0 ;zero index of writing
306     ldi    temp0,HIGH(RS232FIFOBegin) ;set High to begin of buffer
307     sts    RS232ReadPosPtr+1,temp0 ;zero index of reading
308     sts    RS232WritePosPtr+1,temp0 ;zero index of writing
309     sts    RS232LengthPosPtr,YH ;zero index of length
310     sts    RS232LengthPosPtr+1,YH ;zero index of length
311     clr    RS232BufferFull
312
313
314     rcall  InitACKBuffer      ;initialization of ACK buffer
315     rcall  InitNAKBuffer      ;initialization of NAK buffer
316
317     rcall  USBReset           ;initialization of USB addresses
318
319     ldi    temp0,0b00111100   ;set pull-up on PORTB
320     out    PORTB,temp0
321     ldi    temp0,0b11111111   ;set pull-up on PORTC
322     out    PORTC,temp0
323     ldi    temp0,0b11111011   ;set pull-up on PORTD
324     out    PORTD,temp0
325
326     clr    temp0               ;
327     out    UBRRH,temp0         ;set UART speed High
328     out    EEARH,temp0        ;zero EEPROM index
329
330     ldi    temp0,1<<U2X       ;set mode X2 on UART
331     out    USR,temp0
332     ldi    temp0,InitBaudRate  ;set UART speed
333     out    UBRR,temp0
334     sbi    UCR,TXEN            ;enable transmitting of UART
335     sbi    UCR,RXEN            ;enable receiving of UART
336     sbi    UCR,RXCIE          ;enable interrupt from receiving of UART
337
338     ldi    temp0,0x0F          ;INT0 - respond to leading edge
339     out    MCUCR,temp0         ;
340     ldi    temp0,1<<INT0       ;enable external interrupt INT0
341     out    GIMSK,temp0
342 ;-----
343 ;*****
344 ;* Main program
345 ;*****
346     sei                                ;enable interrupts globally
347 Main:
348     sbis   inputport,DATAmisus ;waiting till change D- to 0

```

```

349         rjmp    CheckUSBReset          ;and check, if isn't USB reset
350
351         cpi      ActionFlag,DoReceiveSetupData
352         breq     ProcReceiveSetupData
353         cpi      ActionFlag,DoPrepareOutContinuousBuffer
354         breq     ProcPrepareOutContinuousBuffer
355         rjmp     Main
356
357 CheckUSBReset:
358         ldi      temp0,255                ;counter duration of reset (according to specification is that cca 10ms - here is cca 100us)
359 WaitForUSBReset:
360         sbic     inputport,DATAminus      ;waiting till change D+ to 0
361         rjmp     Main
362         dec      temp0
363         brne     WaitForUSBReset
364         rcall    USBReset
365         rjmp     Main
366
367 ProcPrepareOutContinuousBuffer:
368         rcall    PrepareOutContinuousBuffer ;prepare next sequence of answer to buffer
369         ldi      ActionFlag,DoReadySendAnswer
370         rjmp     Main
371 ProcReceiveSetupData:
372         ldi      USBBufptrY,InputBufferBegin ;pointer to begin of receiving buffer
373         mov      ByteCount,InputBufferLength ;length of input buffer
374         rcall    DecodeNRZI                ;transfer NRZI coding to bits
375         rcall    MirrorInBufferBytes       ;invert bits order in bytes
376         rcall    BitStuff                  ;removal of bitstuffing
377         ;rcall   CheckCRCIn                ;check CRC
378         rcall    PrepareUSBOutAnswer       ;prepare answers to transmitting buffer
379         ldi      ActionFlag,DoReadySendAnswer
380         rjmp     Main
381 ;*****
382 ;* Main program END
383 ;*****
384 ;-----
385 ;*****
386 ;* Interrupt0 interrupt handler
387 ;*****
388 INT0Handler:                                ;interrupt INT0
389         in       backupSREG,SREG
390         push     temp0
391         push     temp1
392
393         ldi      temp0,3                    ;counter of duration log0
394         ldi      temp1,2                    ;counter of duration log1
395         ;waiting for begin packet
396 CheckchangeMinus:
397         sbis     inputport,DATAminus      ;waiting till change D- to 1
398         rjmp     CheckchangeMinus
399 CheckchangePlus:
400         sbis     inputport,DATAplus       ;waiting till change D+ to 1
401         rjmp     CheckchangePlus
402 DetectSOPEnd:
403         sbis     inputport,DATAplus
404         rjmp     Increment0                ;D+ =0
405 Increment1:
406         ldi      temp0,3                    ;counter of duration log0

```

```

407     dec     temp1                ;how many cycles takes log1
408     nop
409     breq    USBBeginPacket       ;if this is end of SOP - receive packet
410     rjmp    DetectSOPEnd
411 Increment0:
412     ldi     temp1,2              ;counter of duration log1
413     dec     temp0                ;how many cycles take log0
414     nop
415     brne    DetectSOPEnd          ;if there isn't SOF - continue
416     rjmp    EndInt0HandlerPOP2
417 EndInt0Handler:
418     pop     ACC
419     pop     RS232BufptrX
420     pop     temp3
421     pop     temp2
422 EndInt0HandlerPOP:
423     pop     USBBufptrY
424     pop     ByteCount
425     mov     bitcount,backupbitcount ;restore bitcount register
426 EndInt0HandlerPOP2:
427     pop     temp1
428     pop     temp0
429     out     SREG,backupSREG
430     ldi     shiftbuf,1<<INTF0    ;zero interruptu flag INTF0
431     out     GIFR,shiftbuf
432     reti
433                                     ;otherwise finish (was only SOF - every millisecond)
434 USBBeginPacket:
435     mov     backupbitcount,bitcount ;backup bitcount register
436     in      shiftbuf,inputport    ;if yes load it as zero bit directly to shift register
437 USBloopBegin:
438     push    ByteCount             ;additional backup of registers (save of time)
439     push    USBBufptrY
440     ldi     bitcount,6            ;initialization of bits counter in byte
441     ldi     ByteCount,MAXUSBBYTES ;initialization of max number of received bytes in packet
442     ldi     USBBufptrY,InputShiftBufferBegin ;set the input buffer
443 USBloop1_6:
444     in      inputbuf,inputport
445     cbr     inputbuf,USBpinmask   ;unmask low 2 bits
446     breq    USBloopEnd           ;if they are zeros - end of USB packet
447     ror     inputbuf             ;transfer Data+ to shift register
448     rol     shiftbuf
449     dec     bitcount             ;decrement bits counter
450     brne    USBloop1_6          ;if it isn't zero - repeat filling of shift register
451     nop                           ;otherwise is necessary copy shift register to buffer
452 USBloop7:
453     in      inputbuf,inputport
454     cbr     inputbuf,USBpinmask   ;unmask low 2 bits
455     breq    USBloopEnd           ;if they are zeros - end of USB packet
456     ror     inputbuf             ;transfer Data+ to shift register
457     rol     shiftbuf
458     ldi     bitcount,7            ;initialization of bits counter in byte
459     st      Y+,shiftbuf          ;copy shift register into buffer and increment pointer to buffer
460 USBloop0:
461     in      shiftbuf,inputport    ;zero bit directly to shift register
462     cbr     shiftbuf,USBpinmask   ;unmask low 2 bits
463     breq    USBloopEnd           ;if they are zeros - end of USB packet
464     dec     bitcount             ;decrement bits counter

```



```

465      nop                                ;
466      dec      ByteCount                 ;if not reached maximum buffer
467      brne     USBloop1_6                ;then receive next
468
469      rjmp      EndInt0HandlerPOP         ;otherwise repeat back from begin
470
471 USBloopEnd:
472      cpi       USBBufptrY,InputShiftBufferBegin+3      ;if at least 3 byte not received
473      brcs      EndInt0HandlerPOP         ;then finish
474      lds       temp0,InputShiftBufferBegin+0           ;identifier of packet to temp0
475      lds       temp1,InputShiftBufferBegin+1           ;address to temp1
476      brne      TestDataPacket            ;if is length different from 3 - then this can be only DataPaket
477 TestIOPacket:
478      ;      cp       temp1,MyAddress         ;if this isn't assigned (address) for me
479      ;      brne     TestDataPacket         ;then this can be still DataPacket
480 TestSetupPacket: ;test to SETUP packet
481      cpi       temp0,nNRZISETUPPID
482      brne      TestOutPacket             ;if this isn't Setup PID - decode other packet
483      cp        temp1,MyInAddress          ;if this isn't assigned (address) for me
484      brne      TestDataPacket            ;then this can be still DataPacket
485      ldi       State,SetupState
486      rjmp      EndInt0HandlerPOP         ;if this is Setup PID - receive consecutive Data packet
487 TestOutPacket: ;test for OUT packet
488      cpi       temp0,nNRZIOUTPID
489      brne      TestInPacket              ;if this isn't Out PID - decode other packet
490      cp        temp1,MyOutAddress         ;if this isn't assigned (address) for me
491      brne      TestDataPacket            ;then this can be still DataPacket
492      ldi       State,OutState
493      rjmp      EndInt0HandlerPOP         ;if this is Out PID - receive consecutive Data packet
494 TestInPacket: ;test on IN packet
495      cpi       temp0,nNRZIINPID
496      brne      TestDataPacket            ;if this isn't In PID - decode other packet
497      cp        temp1,MyInAddress         ;if this isn't assigned (address) for me
498      breq      AnswerToInRequest
499 TestDataPacket: ;test for DATA0 and DATA1 packet
500      cpi       temp0,nNRZIDATA0PID
501      breq      Data0Packet               ;if this isn't Data0 PID - decode other packet
502      cpi       temp0,nNRZIDATA1PID
503      brne      NoMyPacked               ;if this isn't Data1 PID - decode other packet
504 Data0Packet:
505      cpi       State,SetupState          ;if was state Setup
506      breq      ReceiveSetupData          ;receive it
507      cpi       State,OutState            ;if was state Out
508      breq      ReceiveOutData            ;receive it
509 NoMyPacked:
510      ldi       State,BaseState           ;zero state
511      rjmp      EndInt0HandlerPOP         ;and receive consecutive Data packet
512
513 AnswerToInRequest:
514      push      temp2                     ;backup next registers and continue
515      push      temp3
516      push      RS232BufptrX
517      push      ACC
518      cpi       ActionFlag,DoReadySendAnswer ;if isn't prepared answer
519      brne      NoReadySend               ;then send NAK
520      rcall     SendPreparedUSBAnswer      ;transmitting answer back
521      cpi       State,AddressChangeState ;if state is AddressChange
522      breq      SetMyNewUSBAddress         ;then is necessary to change USB address

```

```

523         ldi     State,InState
524         ldi     ActionFlag,DoPrepareOutContinuousBuffer
525         rjmp    EndInt0Handler          ;and repeat - wait for next response from USB
526 ReceiveSetupData:
527         push    temp2                  ;backup next registers and continue
528         push    temp3
529         push    RS232BufptrX
530         push    ACC
531         rcall   SendACK                ;accept Setup Data packet
532         rcall   FinishReceiving        ;finish receiving
533         ldi     ActionFlag,DoReceiveSetupData
534         rjmp    EndInt0Handler
535 ReceiveOutData:
536         push    temp2                  ;backup next registers and continue
537         push    temp3
538         push    RS232BufptrX
539         push    ACC
540         cpi     ActionFlag,DoReceiveSetupData ;if is currently in process command Setup
541         breq    NoReadySend            ;then send NAK
542         rcall   SendACK                ;accept Out packet
543         clr     ActionFlag
544         rjmp    EndInt0Handler
545 NoReadySend:
546         rcall   SendNAK                ;still I am not ready to answer
547         rjmp    EndInt0Handler        ;and repeat - wait for next response from USB
548 ;-----
549 SetMyNewUSBAddress:                  ;set new USB address in NRZI coded
550         lds     MyInAddress,MyInAddressSRAM
551         lds     MyOutAddress,MyOutAddressSRAM
552         rjmp    EndInt0Handler
553 ;-----
554 FinishReceiving:                    ;corrective actions for receive termination
555         cpi     bitcount,7            ;transfer to buffer also last not completed byte
556         breq    NoRemainingBits      ;if were all bytes transfered, then nothing transfer
557         inc     bitcount
558 ShiftRemainingBits:
559         rol     shiftbuf              ;shift remaining not completed bits on right position
560         dec     bitcount
561         brne    ShiftRemainingBits
562         st      Y+,shiftbuf           ;and copy shift register bo buffer - not completed byte
563 NoRemainingBits:
564         mov     ByteCount,USBBufptrY
565         subi    ByteCount,InputShiftBufferBegin-1 ;in ByteCount is number of received bytes (including not completed bytes)
566
567         mov     InputBufferLength,ByteCount ;and save for use in main program
568         ldi     USBBufptrY,InputShiftBufferBegin ;pointer to begin of receiving shift buffer
569         ldi     RS232BufptrX,InputBufferBegin+1 ;data buffer (leave out SOP)
570         push    XH                    ;save RS232BufptrX Hi index
571         clr     XH
572 MoveDataBuffer:
573         ld      temp0,Y+
574         st      X+,temp0
575         dec     ByteCount
576         brne    MoveDataBuffer
577
578         pop     XH                    ;restore RS232BufptrX Hi index
579         ldi     ByteCount,nNRZISOPbyte
580         sts     InputBufferBegin,ByteCount ;like received SOP - it is not copied from shift buffer

```

```

581         ret
582 ;-----
583 ;*****
584 ;* Other procedures
585 ;*****
586 ;-----
587 USBReset:      ;initialization of USB state engine
588         ldi     temp0,nNRZIADDR0      ;initialization of USB address
589         mov     MyOutAddress,temp0
590         mov     MyInAddress,temp0
591         clr     State                  ;initialization of state engine
592         clr     BitStuffInOut
593         clr     OutBitStuffNumber
594         clr     ActionFlag
595         clr     RAMread                ;will be reading from ROM
596         sts     ConfigByte,RAMread    ;unconfigured state
597         ret
598 ;-----
599 SendPreparedUSBAnswer: ;transmitting by NRZI coding OUT buffer with length OutputBufferLength to USB
600         mov     ByteCount,OutputBufferLength ;length of answer
601 SendUSBAnswer:  ;transmitting by NRZI coding OUT buffer to USB
602         ldi     USBBufptrY,OutputBufferBegin ;pointer to begin of transmitting buffer
603 SendUSBBuffer: ;transmitting by NRZI coding given buffer to USB
604         ldi     temp1,0                ;incrementing pointer (temporary variable)
605         mov     temp3,ByteCount        ;byte counter: temp3 = ByteCount
606         ldi     temp2,0b00000011     ;mask for xoring
607         ld      inputbuf,Y+           ;load first byte to inputbuf and increment pointer to buffer
608                                     ;USB as output:
609         cbi     outputport,DATAPLUS   ;down DATAPLUS : idle state of USB port
610         sbi     outputport,DATAMINUS  ;set DATAMINUS : idle state of USB port
611         sbi     USBdirection,DATAPLUS ;DATAPLUS as output
612         sbi     USBdirection,DATAMINUS ;DATAMINUS as output
613
614         in      temp0,outputport      ;idle state of USB port to temp0
615 SendUSBAnswerLoop:
616         ldi     bitcount,7            ;bits counter
617 SendUSBAnswerByteLoop:
618         nop                            ;delay because timing
619         ror     inputbuf               ;to carry transmitting bit (in direction first LSB then MSB)
620         brcs    NoXORSend              ;if that it is one - don't change USB state
621         eor     temp0,temp2            ;otherwise state will be changed
622 NoXORSend:
623         out     outputport,temp0       ;send out to USB
624         dec     bitcount               ;decrement bits counter - according to carry flag
625         brne    SendUSBAnswerByteLoop ;if bits counter isn't zero - repeat transmitting with next bit
626         sbrs    inputbuf,0            ;if is transmitting bit one - don't change USB state
627         eor     temp0,temp2            ;otherwise state will be changed
628 NoXORSendLSB:
629         dec     temp3                 ;decrement bytes counter
630         ld      inputbuf,Y+           ;load next byte and increment pointer to buffer
631         out     outputport,temp0      ;transmit to USB
632         brne    SendUSBAnswerLoop     ;repeat for all buffer (till temp3=0)
633
634         mov     bitcount,OutBitStuffNumber ;bits counter for bitstuff
635         cpi     bitcount,0            ;if not be needed bitstuff
636         breq     ZeroBitStuf
637 SendUSBAnswerBitstuffLoop:
638         ror     inputbuf               ;to carry transmitting bit (in direction first LSB then MSB)

```

```

639      brcs    NoXORBitstuffSend      ;if is one - don't change state on USB
640      eor     temp0,temp2            ;otherwise state will be changed
641 NoXORBitstuffSend:
642      out     outputport,temp0        ;transmit to USB
643      nop                                           ;delay because of timing
644      dec     bitcount                ;decrement bits counter - according to carry flag
645      brne    SendUSBAnswerBitstuffLoop      ;if bits counter isn't zero - repeat transmitting with next bit
646      ld      inputbuf,Y              ;delay 2 cycle
647 ZeroBitStuf:
648      nop                                           ;delay 1 cycle
649      cbr     temp0,3
650      out     outputport,temp0        ;transmit EOP on USB
651
652      ldi     bitcount,5              ;delay counter: EOP should exists 2 bits (16 cycle at 12MHz)
653 SendUSBWaitEOP:
654      dec     bitcount
655      brne    SendUSBWaitEOP
656
657      sbi     outputport,DATAMinus      ;set DATAMINUS : idle state on USB port
658      sbi     outputport,DATAMinus      ;delay 2 cycle: Idle should exists 1 bit (8 cycle at 12MHz)
659      cbi     USBdirection,DATAPlus     ;DATAPLUS as input
660      cbi     USBdirection,DATAMinus    ;DATAMINUS as input
661      cbi     outputport,DATAMinus      ;reset DATAMINUS : the third state on USB port
662      ret
663 ;-----
664 ToggleDATAPID:
665      lds     temp0,OutputBufferBegin+1      ;load last PID
666      cpi     temp0,DATAPID                  ;if last was DATAPID byte
667      ldi     temp0,DATAPID
668      breq    SendData0PID                   ;then send zero answer with DATAPID
669      ldi     temp0,DATAPID                  ;otherwise send zero answer with DATAPID
670 SendData0PID:
671      sts     OutputBufferBegin+1,temp0      ;DATAPID byte
672      ret
673 ;-----
674 ComposeZeroDATAPIDAnswer:
675      ldi     temp0,DATAPID                  ;DATAPID - in the next will be toggled to DATAPID in load descriptor
676      sts     OutputBufferBegin+1,temp0      ;load to output buffer
677 ComposeZeroAnswer:
678      ldi     temp0,SOPbyte
679      sts     OutputBufferBegin+0,temp0      ;SOP byte
680      rcall   ToggleDATAPID                  ;change DATAPID
681      ldi     temp0,0x00
682      sts     OutputBufferBegin+2,temp0      ;CRC byte
683      sts     OutputBufferBegin+3,temp0      ;CRC byte
684      ldi     ByteCount,2+2                  ;length of output buffer (SOP and PID + CRC16)
685      ret
686 ;-----
687 InitACKBuffer:
688      ldi     temp0,SOPbyte
689      sts     ACKBufferBegin+0,temp0          ;SOP byte
690      ldi     temp0,ACKPID
691      sts     ACKBufferBegin+1,temp0          ;ACKPID byte
692      ret
693 ;-----
694 SendACK:
695      push    USBBufptrY
696      push    bitcount

```

```

697      push    OutBitStuffNumber
698      ldi      USBBufptrY,ACKBufferBegin    ;pointer to begin of ACK buffer
699      ldi      ByteCount,2                  ;number of transmit bytes (only SOP and ACKPID)
700      clr      OutBitStuffNumber
701      rcall    SendUSBBuffer
702      pop      OutBitStuffNumber
703      pop      bitcount
704      pop      USBBufptrY
705      ret
706 ;-----
707 InitNAKBuffer:
708      ldi      temp0,SOPbyte
709      sts      NAKBufferBegin+0,temp0        ;SOP byte
710      ldi      temp0,NAKPID
711      sts      NAKBufferBegin+1,temp0        ;NAKPID byte
712      ret
713 ;-----
714 SendNAK:
715      push    OutBitStuffNumber
716      ldi      USBBufptrY,NAKBufferBegin    ;pointer to begin of ACK buffer
717      ldi      ByteCount,2                  ;number of transmitted bytes (only SOP and NAKPID)
718      clr      OutBitStuffNumber
719      rcall    SendUSBBuffer
720      pop      OutBitStuffNumber
721      ret
722 ;-----
723 ComposeSTALL:
724      ldi      temp0,SOPbyte
725      sts      OutputBufferBegin+0,temp0      ;SOP byte
726      ldi      temp0,STALLPID
727      sts      OutputBufferBegin+1,temp0      ;STALLPID byte
728      ldi      ByteCount,2
729      ret
730 ;-----
731 DecodeNRZI:    ;encoding of buffer from NRZI code to binary
732      push    USBBufptrY    ;back up pointer to buffer
733      push    ByteCount     ;back up length of buffer
734      add     ByteCount,USBBufptrY ;end of buffer to ByteCount
735      ser     temp0         ;to ensure unit carry (in the next rotation)
736 NRZIloop:
737      ror     temp0         ;filling carry from previous byte
738      ld      temp0,Y       ;load received byte from buffer
739      mov     temp2,temp0   ;shifted register to one bit to the right and XOR for function of NRZI decoding
740      ror     temp2         ;carry to most significant digit bit and shift
741      eor     temp2,temp0   ;NRZI decoding
742      com     temp2         ;negate
743      st      Y+,temp2      ;save back as decoded byte and increment pointer to buffer
744      cp      USBBufptrY,ByteCount ;if not all bytes
745      brne    NRZIloop      ;then repeat
746      pop     ByteCount     ;restore buffer length
747      pop     USBBufptrY    ;restore pointer to buffer
748      ret
749 ;-----
750 BitStuff:;removal of bitstuffing in buffer
751      clr     temp3         ;counter of omitted bits
752      clr     lastBitstufNumber ;0xFF to lastBitstufNumber
753      dec     lastBitstufNumber
754 BitStuffRepeat:

```

```

755      push    USBBufptrY      ;back up pointer to buffer
756      push    ByteCount      ;back up buffer length
757      mov     temp1,temp3     ;counter of all bits
758      ldi     temp0,8         ;sum all bits in buffer
759 SumAllBits:
760      add     temp1,temp0
761      dec     ByteCount
762      brne    SumAllBits
763      ldi     temp2,6         ;initialize counter of ones
764      pop     ByteCount       ;restore buffer length
765      push    ByteCount       ;back up buffer length
766      add     ByteCount,USBBufptrY ;end of buffer to ByteCount
767      inc     ByteCount       ;and for safety increment it with 2 (because of shifting)
768      inc     ByteCount
769 BitStuffLoop:
770      ld      temp0,Y         ;load received byte from buffer
771      ldi     bitcount,8      ;bits counter in byte
772 BitStuffByteLoop:
773      ror     temp0           ;filling carry from LSB
774      brcs    IncrementBitstuff ;if that LSB=0
775      ldi     temp2,7         ;initialize counter of ones +1 (if was zero)
776 IncrementBitstuff:
777      dec     temp2           ;decrement counter of ones (assumption of one bit)
778      brne    DontShiftBuffer ;if there was not 6 ones together - don't shift buffer
779      cp      temp1,lastBitstufNumber ;
780      ldi     temp2,6         ;initialize counter of ones (if no bitstuffing will be made then must be started again)
781      brcc    DontShiftBuffer ;if already was made bitstuffing - don't shift buffer
782
783      dec     temp1 ;
784      mov     lastBitstufNumber,temp1 ;remember last position of bitstuffing
785      cpi     bitcount,1      ;for pointing to 7-th bit (which must be deleted or where to insert zero)
786      brne    NoBitcountCorrect
787      ldi     bitcount,9 ;
788      inc     USBBufptrY      ;zvys pointer do buffera      ENG;increment pointer to buffer
789 NoBitcountCorrect:
790      dec     bitcount
791      bst     BitStuffInOut,0
792      brts    CorrectOutBuffer ;if this is Out buffer - increment buffer length
793      rcall   ShiftDeleteBuffer ;shift In buffer
794      dec     temp3           ;decrement counter of omission
795      rjmp    CorrectBufferEnd
796 CorrectOutBuffer:
797      rcall   ShiftInsertBuffer ;shift Out buffer
798      inc     temp3           ;increment counter of omission
799 CorrectBufferEnd:
800      pop     ByteCount       ;restore buffer length
801      pop     USBBufptrY      ;restore pointer to buffer
802      rjmp    BitStuffRepeat  ;and restart from begin
803 DontShiftBuffer:
804      dec     temp1           ;if already were all bits
805      breq    EndBitStuff     ;finish cycle
806      dec     bitcount        ;decrement bits counter in byte
807      brne    BitStuffByteLoop ;if not yet been all bits in byte - go to next bit
808      ;otherwise load next byte
809      inc     USBBufptrY      ;increment pointer to buffer
810      rjmp    BitStuffLoop    ;and repeat
811 EndBitStuff:
812      pop     ByteCount       ;restore buffer length

```

```

813     pop     USBBufptrY      ;restore pointer to buffer
814     bst     BitStuffInOut,0
815     brts    IncrementLength ;if this is Out buffer - increment length of Out buffer
816 DecrementLength:           ;if this is In buffer - decrement length of In buffer
817     cpi     temp3,0         ;was at least one decrement
818     breq    NoChangeByteCount ;if no - don't change buffer length
819     dec     ByteCount       ;if this is In buffer - decrement buffer length
820     subi    temp3,256-8     ;if there wasn't above 8 bits over
821     brcc    NoChangeByteCount ;then finish
822     dec     ByteCount       ;otherwise next decrement buffer length
823     ret                                           ;and finish
824 IncrementLength:
825     mov     OutBitStuffNumber,temp3 ;remember number of bits over
826     subi    temp3,8         ;if there wasn't above 8 bits over
827     brcs    NoChangeByteCount ;then finish
828     inc     ByteCount       ;otherwise increment buffer length
829     mov     OutBitStuffNumber,temp3 ;and remember number of bits over (decremented by 8)
830 NoChangeByteCount:
831     ret                                           ;finish
832 ;-----
833 ShiftInsertBuffer:         ;shift buffer by one bit to right from end till to position: byte-USBBufptrY and bit-bitcount
834     mov     temp0,bitcount ;calculation: bitcount= 9-bitcount
835     ldi     bitcount,9
836     sub     bitcount,temp0 ;to bitcount bit position, which is necessary to clear
837
838     ld      temp1,Y         ;load byte which still must be shifted from position bitcount
839     rol     temp1          ;and shift to the left through Carry (transmission from higher byte and LSB to Carry)
840     ser     temp2          ;FF to mask - temp2
841 HalfInsertPosuvMask:
842     lsl     temp2          ;zero to the next low bit of mask
843     dec     bitcount       ;till not reached boundary of shifting in byte
844     brne    HalfInsertPosuvMask
845
846     and     temp1,temp2    ;unmask that remains only high shifted bits in temp1
847     com     temp2          ;invert mask
848     lsr     temp2          ;shift mask to the right - for insertion of zero bit
849     ld      temp0,Y         ;load byte which must be shifted from position bitcount to temp0
850     and     temp0,temp2    ;unmask to remains only low non-shifted bits in temp0
851     or      temp1,temp0    ;and put together shifted and nonshifted part
852
853     ld      temp0,Y         ;load byte which must be shifted from position bitcount
854     rol     temp0          ;and shift it to the left through Carry (to set right Carry for further carry)
855     st      Y+,temp1       ;and load back modified byte
856 ShiftInsertBufferLoop:
857     cpse    USBBufptrY,ByteCount ;if are not all entire bytes
858     rjmp    NoEndShiftInsertBuffer ;then continue
859     ret                                           ;otherwise finish
860 NoEndShiftInsertBuffer:
861     ld      temp1,Y         ;load byte
862     rol     temp1          ;and shift to the left through Carry (carry from low byte and LSB to Carry)
863     st      Y+,temp1       ;and store back
864     rjmp    ShiftInsertBufferLoop ;and continue
865 ;-----
866 ShiftDeleteBuffer:         ;shift buffer one bit to the left from end to position: byte-USBBufptrY and bit-bitcount
867     mov     temp0,bitcount ;calculation: bitcount= 9-bitcount
868     ldi     bitcount,9
869     sub     bitcount,temp0 ;to bitcount bit position, which must be shifted
870     mov     temp0,USBBufptrY ;backup pointer to buffer

```



```

871         inc     temp0                ;position of completed bytes to temp0
872         mov     USBBufptrY,ByteCount ;maximum position to pointer
873 ShiftDeleteBufferLoop:
874         ld      temp1,-Y              ;decrement buffer and load byte
875         ror     temp1                 ;and right shift through Carry (carry from higher byte and LSB to Carry)
876         st      Y,temp1              ;and store back
877         cpse    USBBufptrY,temp0     ;if there are not all entire bytes
878         rjmp     ShiftDeleteBufferLoop ;then continue
879
880         ld      temp1,-Y              ;decrement buffer and load byte which must be shifted from position bitcount
881         ror     temp1                 ;and right shift through Carry (carry from higher byte and LSB to Carry)
882         ser     temp2                 ;FF to mask - temp2
883 HalfDeletePosuvMask:
884         dec     bitcount              ;till not reached boundary of shifting in byte
885         breq     DoneMask
886         lsl     temp2                 ;zero to the next low bit of mask
887         rjmp     HalfDeletePosuvMask
888 DoneMask:
889         and     temp1,temp2           ;unmask to remain only high shifted bits in temp1
890         com     temp2                 ;invert mask
891         ld      temp0,Y               ;load byte which must be shifted from position bitcount to temp0
892         and     temp0,temp2           ;unmask to remain only low nonshifted bits in temp0
893         or      temp1,temp0           ;and put together shifted and nonshifted part
894         st      Y,temp1              ;and store back
895         ret                          ;and finish
896 ;-----
897 MirrorInBufferBytes:
898         push    USBBufptrY
899         push    ByteCount
900         ldi     USBBufptrY,InputBufferBegin
901         rcall   MirrorBufferBytes
902         pop     ByteCount
903         pop     USBBufptrY
904         ret
905 ;-----
906 MirrorBufferBytes:
907         add     ByteCount,USBBufptrY ;ByteCount shows to the end of message
908 MirrorBufferloop:
909         ld      temp0,Y               ;load received byte from buffer
910         ldi     temp1,8               ;bits counter
911 MirrorBufferByteLoop:
912         ror     temp0                 ;to carry next least bit
913         rol     temp2                 ;from carry next bit to reverse order
914         dec     temp1                 ;was already entire byte
915         brne    MirrorBufferByteLoop ;if no then repeat next least bit
916         st      Y+,temp2              ;save back as reversed byte and increment pointer to buffer
917         cp      USBBufptrY,ByteCount ;if not yet been all
918         brne    MirrorBufferloop     ;then repeat
919         ret                          ;otherwise finish
920 ;-----
921 ;CheckCRCIn:
922 ;         kiss    USBBUFPTRY
923 ;         kiss    ByteCount
924 ;         ldi     USBBUFPTRY,InputBuffercompare
925 ;         rcall   CheckCRC
926 ;         pope    ByteCount
927 ;         pope    USBBUFPTRY
928 ;         lip

```



```

929 ;-----
930 AddCRCOut:
931     push    USBBufptrY
932     push    ByteCount
933     ldi     USBBufptrY,OutputBufferBegin
934     rcall   CheckCRC
935     com     temp0                ;negation of CRC
936     com     temp1
937     st      Y+,temp1            ;save CRC to the end of buffer (at first MSB)
938     st      Y,temp0            ;save CRC to the end of buffer (then LSB)
939     dec     USBBufptrY          ;pointer to CRC position
940     ldi     ByteCount,2        ;reverse bits order in 2 bytes CRC
941     rcall   MirrorBufferBytes  ;reverse bits order in CRC (transmitting CRC - MSB first)
942     pop     ByteCount
943     pop     USBBufptrY
944     ret
945 ;-----
946 CheckCRC:;input: USBBufptrY = begin of message ,ByteCount = length of message
947     add     ByteCount,USBBufptrY ;ByteCount points to the end of message
948     inc     USBBufptrY          ;set the pointer to message start - omit SOP
949     ld      temp0,Y+           ;load PID to temp0
950                                     ;and set the pointer to start of message - omit also PID
951     cpi     temp0,DATA0PID      ;if is DATA0 field
952     breq    ComputedATACRC      ;compute CRC16
953     cpi     temp0,DATA1PID      ;if is DATA1 field
954     brne    CRC16End           ;if no then finish
955 ComputedATACRC:
956     ser     temp0              ;initialization of remainder LSB to 0xff
957     ser     temp1              ;initialization of remainder MSB to 0xff
958 CRC16Loop:
959     ld      temp2,Y+           ;load message to temp2 and increment pointer to buffer
960     ldi     temp3,8            ;bits counter in byte - temp3
961 CRC16LoopByte:
962     bst     temp1,7            ;to T save MSB of remainder (remainder is only 16 bits - 8 bit of higher byte)
963     bld     bitcount,0         ;to bitcount LSB save T - of MSB remainder
964     eor     bitcount,temp2     ;XOR of bit message and bit remainder - in LSB bitcount
965     rol     temp0              ;shift remainder to the left - low byte (two bytes - through carry)
966     rol     temp1              ;shift remainder to the left - high byte (two bytes - through carry)
967     cbr     temp0,1            ;znuluj LSB remains
968     lsr     temp2              ;shift message to right
969     ror     bitcount           ;result of XOR bits from LSB to carry
970     brcc    CRC16NoXOR         ;if is XOR bitmessage and MSB of remainder = 0 , then no XOR
971     ldi     bitcount,CRC16poly>>8 ;to bitcount CRC polynomial - high byte
972     eor     temp1,bitcount      ;and make XOR from remains and CRC polynomial - high byte
973     ldi     bitcount,CRC16poly ;to bitcount CRC polynomial - low byte
974     eor     temp0,bitcount      ;and make XOR of remainder and CRC polynomial - low byte
975 CRC16NoXOR:
976     dec     temp3              ;were already all bits in byte
977     brne    CRC16LoopByte      ;unless, then go to next bit
978     cp      USBBufptrY,ByteCount ;was already end-of-message
979     brne    CRC16Loop          ;unless then repeat
980 CRC16End:
981     ret                        ;otherwise finish (in temp0 and temp1 is result)
982 ;-----
983 LoadDescriptorFromROM:
984     lpm     ;load from ROM position pointer to R0
985     st      Y+,R0              ;R0 save to buffer and increment buffer
986     adiw    ZH:ZL,1            ;increment index to ROM

```

```

987      dec    ByteCount          ;till are not all bytes
988      brne   LoadDescriptorFromROM ;then load next
989      rjmp   EndFromRAMROM      ;otherwise finish
990  ;-----
991  LoadDescriptorFromROMZeroInsert:
992      lpm          ;load from ROM position pointer to R0
993      st         Y+,R0      ;R0 save to buffer and increment buffer
994
995      bst        RAMread,3    ;if bit 3 is one - don't insert zero
996      brtc       InsertingZero ;otherwise zero will be inserted
997      adiw       ZH:ZL,1      ;increment index to ROM
998      lpm          ;load from ROM position pointer to R0
999      st         Y+,R0      ;R0 save to buffer and increment buffer
1000     clt         ;and clear
1001     bld        RAMread,3    ;the third bit in RAMread - for to the next zero insertion will be made
1002     rjmp       InsertingZeroEnd ;and continue
1003  InsertingZero:
1004     clr        R0          ;for insertion of zero
1005     st         Y+,R0      ;zero save to buffer and increment buffer
1006  InsertingZeroEnd:
1007     adiw       ZH:ZL,1      ;increment index to ROM
1008     subi       ByteCount,2  ;till are not all bytes
1009     brne       LoadDescriptorFromROMZeroInsert ;then load next
1010     rjmp       EndFromRAMROM ;otherwise finish
1011  ;-----
1012  LoadDescriptorFromSRAM:
1013     ld         R0,Z        ;load from position RAM pointer to R0
1014     st         Y+,R0      ;R0 save to buffer and increment buffer
1015     adiw       ZH:ZL,1      ;increment index to RAM
1016     dec        ByteCount    ;till are not all bytes
1017     brne       LoadDescriptorFromSRAM ;then load next
1018     rjmp       EndFromRAMROM ;otherwise finish
1019  ;-----
1020  LoadDescriptorFromEEPROM:
1021     out        EEARL,ZL    ;set the address EEPROM Lo
1022     out        EEARH,ZH    ;set the address EEPROM Hi
1023     sbi        EECR,EERE   ;read EEPROM to register EEDR
1024     in         R0,EEDR     ;load from EEDR to R0
1025     st         Y+,R0      ;R0 save to buffer and increment buffer
1026     adiw       ZH:ZL,1      ;increment index to EEPROM
1027     dec        ByteCount    ;till are not all bytes
1028     brne       LoadDescriptorFromEEPROM ;then load next
1029     rjmp       EndFromRAMROM ;otherwise finish
1030  ;-----
1031  LoadXXXDescriptor:
1032     ldi        temp0,SOPbyte ;SOP byte
1033     sts        OutputBufferBegin,temp0 ;to begin of transmitting buffer store SOP
1034     ldi        ByteCount,8   ;8 byte store
1035     ldi        USBBufptrY,OutputBufferBegin+2 ;to transmitting buffer
1036
1037     and        RAMread,RAMread ;if will be reading from RAM or ROM or EEPROM
1038     brne       FromRAMorEEPROM ;0=ROM,1=RAM,2=EEPROM,4=ROM with zero insertion (string)
1039  FromROM:
1040     rjmp       LoadDescriptorFromROM ;load descriptor from ROM
1041  FromRAMorEEPROM:
1042     sbrc       RAMread,2      ;if RAMREAD=4
1043     rjmp       LoadDescriptorFromROMZeroInsert ;read from ROM with zero insertion
1044     sbrc       RAMread,0      ;if RAMREAD=1

```

```

1045      rjmp    LoadDescriptorFromSRAM      ;load data from SRAM
1046      rjmp    LoadDescriptorFromEEPROM    ;otherwise read from EEPROM
1047 EndFromRAMROM:
1048      sbrc    RAMread,7                    ;if is most significant bit in variable RAMread=1
1049      clr     RAMread                      ;clear RAMread
1050      rcall   ToggleDATAPID               ;change DATAPID
1051      ldi     USBBufptrY,OutputBufferBegin+1 ;to transmitting buffer - position of DATA PID
1052      ret
1053 ;-----
1054 PrepareUSBOutAnswer: ;prepare answer to buffer
1055      rcall   PrepareUSBAnswer             ;prepare answer to buffer
1056 MakeOutBitStuff:
1057      inc     BitStuffInOut                ;transmitting buffer - insertion of bitstuff bits
1058      ldi     USBBufptrY,OutputBufferBegin ;to transmitting buffer
1059      rcall   BitStuff
1060      mov     OutputBufferLength,ByteCount ;length of answer store for transmitting
1061      clr     BitStuffInOut                ;receiving buffer - deletion of bitstuff bits
1062      ret
1063 ;-----
1064 PrepareUSBAnswer: ;prepare answer to buffer
1065      clr     RAMread                      ;zero to RAMread variable - reading from ROM
1066      lds     temp0,InputBufferBegin+2     ;bmRequestType to temp0
1067      lds     temp1,InputBufferBegin+3     ;bRequest to temp1
1068      cbr     temp0,0b10011111            ;if is 5 and 6 bit zero
1069      brne    VendorRequest               ;then this isn't Vendor Request
1070      rjmp    StandardRequest              ;but this is standard Request
1071 ;-----
1072 VendorRequest:
1073      clr     ZH                           ;for reading from RAM or EEPROM
1074
1075      cpi     temp1,1                       ;
1076      brne    NoDoSetInfraBufferEmpty      ;
1077      rjmp    DoSetInfraBufferEmpty         ;restart infra receiving (if it was stopped by reading from RAM)
1078 NoDoSetInfraBufferEmpty:
1079      cpi     temp1,2                       ;
1080      brne    NoDoGetInfraCode             ;
1081      rjmp    DoGetInfraCode                ;transmit received infra code (if it is in buffer)
1082 NoDoGetInfraCode:
1083      cpi     temp1,3                       ;
1084      brne    NoDoSetDataPortDirection     ;
1085      rjmp    DoSetDataPortDirection        ;set flow direction of datal bits
1086 NoDoSetDataPortDirection:
1087      cpi     temp1,4                       ;
1088      brne    NoDoGetDataPortDirection     ;
1089      rjmp    DoGetDataPortDirection        ;detect of flow direction of data bits
1090 NoDoGetDataPortDirection:
1091      cpi     temp1,5                       ;
1092      brne    NoDoSetOutDataPort           ;
1093      rjmp    DoSetOutDataPort              ;set data bits (if they are inputs, then pull-ups)
1094 NoDoSetOutDataPort:
1095      cpi     temp1,6                       ;
1096      brne    NoDoGetOutDataPort           ;
1097      rjmp    DoGetOutDataPort              ;detect settings of data out bits (if they are input, then pull-ups)
1098 NoDoGetOutDataPort:
1099      cpi     temp1,7                       ;
1100      brne    NoDoGetInDataPort            ;
1101      rjmp    DoGetInDataPort               ;return value of input data port
1102 NoDoGetInDataPort:

```

```
1103         cpi      temp1,8                ;
1104         brne      NoDoEEPROMRead
1105         rjmp      DoEEPROMRead           ;return contents of EEPROM from given address
1106 NoDoEEPROMRead:
1107         cpi      temp1,9                ;
1108         brne      NoDoEEPROMWrite
1109         rjmp      DoEEPROMWrite         ;write to EEPROM to given address given data
1110 NoDoEEPROMWrite:
1111         cpi      temp1,10               ;
1112         brne      NoDoRS232Send
1113         rjmp      DoRS232Send          ;transmit byte to serial line
1114 NoDoRS232Send:
1115         cpi      temp1,11               ;
1116         brne      NoDoRS232Read
1117         rjmp      DoRS232Read          ;returns received byte from serial line
1118 NoDoRS232Read:
1119         cpi      temp1,12               ;
1120         brne      NoDoSetRS232Baud
1121         rjmp      DoSetRS232Baud       ;set line speed of of serial line
1122 NoDoSetRS232Baud:
1123         cpi      temp1,13               ;
1124         brne      NoDoGetRS232Baud
1125         rjmp      DoGetRS232Baud      ;return line speed of serial line
1126 NoDoGetRS232Baud:
1127         cpi      temp1,14               ;
1128         brne      NoDoGetRS232Buffer
1129         rjmp      DoGetRS232Buffer    ;return line speed of serial line
1130 NoDoGetRS232Buffer:
1131         cpi      temp1,15               ;
1132         brne      NoDoSetRS232DataBits
1133         rjmp      DoSetRS232DataBits  ;set line speed of serial line
1134 NoDoSetRS232DataBits:
1135         cpi      temp1,16               ;
1136         brne      NoDoGetRS232DataBits
1137         rjmp      DoGetRS232DataBits  ;return line speed of serial line
1138 NoDoGetRS232DataBits:
1139         cpi      temp1,17               ;
1140         brne      NoDoSetRS232Parity
1141         rjmp      DoSetRS232Parity    ;set line speed of serial line
1142 NoDoSetRS232Parity:
1143         cpi      temp1,18               ;
1144         brne      NoDoGetRS232Parity
1145         rjmp      DoGetRS232Parity    ;return line speed of serial line
1146 NoDoGetRS232Parity:
1147         cpi      temp1,19               ;
1148         brne      NoDoSetRS232StopBits
1149         rjmp      DoSetRS232StopBits  ;set line speed of serial line
1150 NoDoSetRS232StopBits:
1151         cpi      temp1,20               ;
1152         brne      NoDoGetRS232StopBits
1153         rjmp      DoGetRS232StopBits  ;return line speed of serial line
1154 NoDoGetRS232StopBits:
1155
1156         cpi      temp1,USER_FNC_NUMBER+0 ;
1157         brne      NoDoUserFunction0
1158         rjmp      DoUserFunction0      ;execute of user function0
1159 NoDoUserFunction0:
1160         cpi      temp1,USER_FNC_NUMBER+1 ;
```

```

1161         brne    NoDoUserFunction1
1162         rjmp     DoUserFunction1           ;execute of user function1
1163 NoDoUserFunction1:
1164         cpi      temp1,USER_FNC_NUMBER+2   ;
1165         brne     NoDoUserFunction2
1166         rjmp     DoUserFunction2           ;execute of user function1
1167 NoDoUserFunction2:
1168
1169         rjmp     ZeroDATA1Answer           ;if that it was something unknown, then prepare zero answer
1170
1171
1172 ;----- USER FUNCTIONS -----
1173
1174 ;-----TEMPLATE OF YOUR FUNCTION-----
1175 ;----- BEGIN: This is template how to write own function -----
1176
1177 ;free of use are registers:
1178 ;temp0,temp1,temp2,temp3,ACC,ZH,ZL
1179 ;registers are destroyed after execution (use push/pop to save content)
1180
1181 ;at the end of routine you must correctly set registers:
1182 ;RAMread - 0=reading from ROM, 1=reading from RAM, 2=reading from EEPROM
1183 ;temp0 - number of transmitted data bytes
1184 ;ZH,ZL - pointer to buffer of transmitted data (pointer to ROM/RAM/EEPROM)
1185
1186 ;to transmit data (preparing data to buffer) :
1187 ;to transmit data you must jump to "ComposeEndXXXDescriptor"
1188 ;to transmit one zero byte you can jump to "OneZeroAnswer" (commonly used as confirmation of correct processing)
1189 ;to transmit two zero byte you can jump to "TwoZeroAnswer" (commonly used as confirmation of error in processing)
1190 ;for small size (up to 8 bytes) answer use buffer AnswerArray (see function DoGetOutDataPort:)
1191
1192 DoUserFunctionX:
1193 DoUserFunction0: ;send byte(s) of RAM starting at position given by first parameter in function
1194         lds      temp0,InputBufferBegin+4   ;first parameter Lo into temp0
1195         lds      temp1,InputBufferBegin+5   ;first parameter Hi into temp1
1196         lds      temp2,InputBufferBegin+6   ;second parameter Lo into temp2
1197         lds      temp3,InputBufferBegin+7   ;second parameter Hi into temp3
1198         lds      ACC,InputBufferBegin+8     ;number of requested bytes from USB host (computer) into ACC
1199
1200         ;Here add your own code:
1201         ;-----
1202         nop                                ;example of code - nothing to do
1203         nop
1204         nop
1205         nop
1206         nop
1207         ;-----
1208
1209         mov      ZL,temp0                   ;will be sending value of RAM - from address stored in temp0 (first parameter Lo of function)
1210         mov      ZH,temp1                   ;will be sending value of RAM - from address stored in temp1 (first parameter Hi of function)
1211         inc      RAMread                     ;RAMread=1 - reading from RAM
1212         ldi      temp0,255                   ;send max number of bytes - 255 bytes are maximum
1213         rjmp     ComposeEndXXXDescriptor    ;a prepare data
1214 DoUserFunction1:
1215         rjmp     OneZeroAnswer              ;only confirm receiving by one zero byte answer
1216 DoUserFunction2:
1217         rjmp     TwoZeroAnswer              ;only confirm receiving by two zero bytes answer
1218 ;----- END: This is template how to write own function -----

```

```

1219
1220
1221 ;----- USER FUNCTIONS -----
1222 ;-----
1223 DoSetInfraBufferEmpty:
1224     rjmp     OneZeroAnswer           ;acknowledge reception with single zero
1225 ;-----
1226 DoGetInfraCode:
1227     rjmp     OneZeroAnswer           ;acknowledge reception with single zero
1228 ;-----
1229 DoSetDataPortDirection:
1230     lds      temp1,InputBufferBegin+7 ;fourth parameter - bit mask - which port(s) to change
1231
1232     lds      temp0,InputBufferBegin+4 ;first parameter - direction of data bits DDRB
1233     andi     temp0,0b00111100         ;mask unused pins
1234     sbrc     temp1,0                 ;if bit0 is zero - don't change port state
1235     out      DDRB,temp0              ;and update direction of data port
1236
1237     lds      temp0,InputBufferBegin+5 ;second parameter - direction of data bits DDRC
1238     sbrc     temp1,1                 ;if bit1 is zero - don't change port state
1239     out      DDRC,temp0              ;and update direction of data port
1240
1241     lds      temp0,InputBufferBegin+6 ;third parameter - direction of data bits DDRD
1242     andi     temp0,0b11111000         ;mask unused pins
1243     ori      temp0,0b00000010        ;mask unused pins
1244     sbrc     temp1,2                 ;if bit2 is zero - don't change port state
1245     out      DDRD,temp0              ;and update direction of data port
1246
1247     rjmp     OneZeroAnswer           ;acknowledge reception with single zero
1248 ;-----
1249 DoGetDataPortDirection:
1250     in        temp0,DDRB              ;read direction of DDRB
1251     sts       AnswerArray,temp0       ;to array AnswerArray
1252     in        temp0,DDRC              ;read direction of DDRC
1253     sts       AnswerArray+1,temp0     ;to array AnswerArray
1254     in        temp0,DDRD              ;read direction of DDRD
1255     sts       AnswerArray+2,temp0     ;to array AnswerArray
1256     ldi       ZL,AnswerArray          ;sending is value from AnswerArray
1257     ldi       temp0,0x81              ;RAMREAD=1 - reading from RAM
1258     mov       RAMread,temp0           ;(highest bit set to 1 - to zero RAMread immediatelly)
1259     ldi       temp0,3                 ;sending are three bytes
1260     rjmp      ComposeEndXXXDescriptor ;and prepare data
1261 ;-----
1262 DoSetOutDataPort:
1263     lds      temp1,InputBufferBegin+7 ;fourth parameter - bit mask - which port(s) to change
1264
1265     lds      temp0,InputBufferBegin+4 ;first parameter - value of data bits PORTB
1266     andi     temp0,0b00111100         ;mask unused pins
1267     sbrc     temp1,0                 ;if bit0 is zero - don't change port state
1268     out      PORTB,temp0             ;and update data port
1269
1270     lds      temp0,InputBufferBegin+5 ;second parameter - value of data bits PORTC
1271     sbrc     temp1,1                 ;if bit1 is zero - don't change port state
1272     out      PORTC,temp0             ;and update data port
1273
1274     lds      temp0,InputBufferBegin+6 ;third parameter - value of data bits PORTD
1275     andi     temp0,0b11111000         ;mask unused pins
1276     ori      temp0,0b00000011        ;mask unused pins

```

```

1277      sbrc      temp1,2           ;if bit2 is zero - don't change port state
1278      out       PORTD,temp0       ;and update data port
1279
1280      rjmp      OneZeroAnswer     ;acknowledge reception with single zero
1281 ;-----
1282 DoGetOutDataPort:
1283      in        temp0,PORTB       ;read PORTB
1284      sts       AnswerArray,temp0 ;to array AnswerArray
1285      in        temp0,PORTC       ;read PORTC
1286      sts       AnswerArray+1,temp0 ;to array AnswerArray
1287      in        temp0,PORTD       ;read PORTD
1288      sts       AnswerArray+2,temp0 ;to array AnswerArray
1289      ldi       ZL,AnswerArray    ;sending is value from AnswerArray
1290      ldi       temp0,0x81        ;RAMREAD=1 - reading from RAM
1291      mov       RAMread,temp0     ;(highest bit set to 1 - to zero RAMread immediatelly)
1292      ldi       temp0,3          ;sending are three bytes
1293      rjmp      ComposeEndXXXDescriptor ;and prepare data
1294 ;-----
1295 DoGetInDataPort:
1296      in        temp0,PINB       ;read PINB
1297      sts       AnswerArray,temp0 ;to array AnswerArray
1298      in        temp0,PINC       ;read PINC
1299      sts       AnswerArray+1,temp0 ;to array AnswerArray
1300      in        temp0,PIND       ;read PIND
1301      sts       AnswerArray+2,temp0 ;to array AnswerArray
1302      ldi       ZL,AnswerArray    ;sending is value from AnswerArray
1303      ldi       temp0,0x81        ;RAMREAD=1 - reading from RAM
1304      mov       RAMread,temp0     ;(highest bit set to 1 - to zero RAMread immediatelly)
1305      ldi       temp0,3          ;sending are three bytes
1306      rjmp      ComposeEndXXXDescriptor ;and prepare data
1307 ;-----
1308 DoGetIn:
1309      ldi       ZL,0              ;sending value in R0
1310      ldi       temp0,0x81        ;RAMread=1 - reading from RAM
1311      mov       RAMread,temp0     ;(highest bit set to 1 - to zero RAMread immediatelly)
1312      ldi       temp0,1          ;send only single byte
1313      rjmp      ComposeEndXXXDescriptor ;and prepare data
1314 ;-----
1315 DoEEPROMRead:
1316      lds       ZL,InputBufferBegin+4 ;first parameter - offset in EEPROM
1317      lds       ZH,InputBufferBegin+5
1318      ldi       temp0,2
1319      mov       RAMread,temp0     ;RAMREAD=2 - reading from EEPROM
1320      ldi       temp0,E2END+1     ;number my byte answers to temp0 - entire length of EEPROM
1321      rjmp      ComposeEndXXXDescriptor ;otherwise prepare data
1322 ;-----
1323 DoEEPROMWrite:
1324      lds       ZL,InputBufferBegin+4 ;first parameter - offset in EEPROM (address)
1325      lds       ZH,InputBufferBegin+5
1326      lds       R0,InputBufferBegin+6 ;second parameter - data to store to EEPROM (data)
1327      out       EEAR,ZL           ;set the address of EEPROM
1328      out       EEARH,ZH
1329      out       EEDR,R0          ;set the data to EEPROM
1330      cli       ;disable interrupt
1331      sbi       EECR,EEMWE       ;set the master write enable
1332      sei       ;enable interrupt (next instruction is performed)
1333      sbi       EECR,EWE        ;write
1334 WaitForEEPROMReady:

```



```

1335      sbic      EECR,EWE          ;wait to the end of write
1336      rjmp      WaitForEEPROMReady ;in loop (max cca 4ms) (because of possible next reading/writing)
1337      rjmp      OneZeroAnswer     ;acknowledge reception with single zero
1338      ;-----
1339 DoRS232Send:
1340      lds        temp0,InputBufferBegin+4 ;first parameter - value transmitted to RS232
1341      out        UDR,temp0              ;transmit data to UART
1342      WaitForRS232Send:
1343      sbis        UCR, TXEN            ;if disabled UART transmitter
1344      rjmp      OneZeroAnswer         ;then finish - protection because loop lock in AT90S2323/2343
1345      sbis        USR, TXC             ;wait for transmittion finish
1346      rjmp      WaitForRS232Send
1347      rjmp      OneZeroAnswer         ;acknowledge reception with single zero
1348      ;-----
1349 DoRS232Read:
1350      rjmp      TwoZeroAnswer         ;only acknowledge reception with two zero
1351      ;-----
1352 DoSetRS232Baud:
1353      lds        temp0,InputBufferBegin+4 ;first parameter - value of baudrate of RS232
1354      lds        temp1,InputBufferBegin+6 ;second parameter - baudrate of RS232 - high byte
1355      cbr        temp1,1<<URSEL        ;writing will be baudrate high byte (no UCSRC)
1356      out        UBRRH,temp1           ;set the speed of UART high byte
1357      out        UBRR,temp0            ;set the speed of UART low byte
1358      rjmp      OneZeroAnswer         ;acknowledge reception with single zero
1359      ;-----
1360 DoGetRS232Baud:
1361      in         temp0,UBRR            ;return speed of UART Lo
1362      sts        AnswerArray,temp0
1363      in         temp0,UBRRH           ;return speed of UART Hi
1364      sts        AnswerArray+1,temp0   ;to array AnswerArray
1365      ldi        ZL,AnswerArray        ;sending is value from AnswerArray
1366      ldi        temp0,0x81            ;RAMREAD=1 - reading from RAM
1367      mov        RAMread,temp0         ;(highest bit set to 1 - to zero RAMread immediatelly)
1368      ldi        temp0,2              ;sending are two bytes
1369      rjmp      ComposeEndXXXDescriptor ;and prepare data
1370      ;-----
1371 DoGetRS232Buffer:
1372      cbi        UCR,RXCIE            ;disable interrupt from UART receiving
1373      nop
1374      lds        temp0,RS232LengthPosPtr
1375      lds        temp1,RS232LengthPosPtr+1 ;obtain buffer length of RS232 code
1376      sbi        UCR,RXCIE            ;enable interrupt from UART receiving
1377
1378      cpi        temp0,0               ;if this isn't RS232 Buffer empty
1379      brne       SomeRS232Send         ;then send it
1380      cpi        temp1,0               ;if this isn't RS232 Buffer empty
1381      brne       SomeRS232Send         ;then send it
1382      rjmp      OneZeroAnswer         ;otherwise nothing send and acknowledge reception with single zero
1383      SomeRS232Send:
1384      lds        ACC,InputBufferBegin+8 ;number of requiring bytes to ACC
1385      ldi        temp2,2               ;number of possible bytes (plus word of buffer length)
1386      add        temp0,temp2
1387      ldi        temp2,0
1388      adc        temp1,temp2
1389      cpi        temp1,0               ;if is MSB>0
1390      brne       AsRequiredGetRS232Buffer ;transmit as many as requested
1391      cp         ACC,temp0             ;if no requested more that I can send
1392      brcc       NoShortGetRS232Buffer ;transmit as many as requested

```



```

1393 AsRequiredGetRS232Buffer:
1394     mov     temp0,ACC
1395     ldi     temp1,0
1396 NoShortGetRS232Buffer:
1397     subi    temp0,2                ;subtract word length
1398     sbci    temp1,0
1399     lds     temp2,RS232ReadPosPtr   ;obtain index of reading of buffer of RS232 code
1400     lds     temp3,RS232ReadPosPtr+1
1401     add     temp2,temp0            ;obtain where is end
1402     adc     temp3,temp1
1403     cpi     temp3,HIGH(RS232BufferEnd+1) ;if it would overflow
1404     brlo    ReadNoOverflow         ;
1405     brne    ReadOverflow           ;if yes - skip to overflow
1406
1407     cpi     temp2,LOW(RS232BufferEnd+1) ;otherwise compare LSB
1408     brlo    ReadNoOverflow         ;and do the same
1409 ReadOverflow:
1410     subi    temp2,LOW(RS232BufferEnd+1) ;caculate how many not transfered
1411     sbci    temp3,HIGH(RS232BufferEnd+1) ;caculate how many not transfered
1412     sub     temp0,temp2            ;and with this short length of reading
1413     sbc     temp1,temp3            ;and with this short length of reading
1414     ldi     temp2,LOW(RS232FIFOBegin) ;and start from zero
1415     ldi     temp3,HIGH(RS232FIFOBegin) ;and start from zero
1416 ReadNoOverflow:
1417     lds     ZL,RS232ReadPosPtr     ;obtain index of reading of buffer of RS232 code
1418     lds     ZH,RS232ReadPosPtr+1   ;obtain index of reading of buffer of RS232 code
1419
1420     sts     RS232ReadPosPtr,temp2   ;write new index of reading of buffer of RS232 code
1421     sts     RS232ReadPosPtr+1,temp3 ;write new index of reading of buffer of RS232 code
1422     sbiw    ZL,2                   ;space for length data - transmitted as first word
1423
1424     cbi     UCR,RXCIE              ;disable interrupt from UART receiving
1425     inc     RAMread                 ;RAMread=1 reading from RAM
1426     lds     temp2,RS232LengthPosPtr
1427     lds     temp3,RS232LengthPosPtr+1 ;obtain buffer length of RS232 code
1428     sub     temp2,temp0            ;decrement buffer length
1429     sbc     temp3,temp1
1430     sts     RS232LengthPosPtr,temp2 ;write new buffer length of RS232 code
1431     sts     RS232LengthPosPtr+1,temp3
1432     sbi     UCR,RXCIE              ;enable interrupt from UART receiving
1433
1434     st      Z+,temp2               ;and save real length to packet
1435     st      Z,temp3               ;and save real length to packet
1436     sbiw    ZL,1                 ;and set to begin
1437     inc     temp0                 ;and about this word increment number of transmited bytes (buffer length)
1438     inc     temp0
1439     rjmp    ComposeEndXXXDescriptor ;and prepare data
1440 ;-----
1441 DoSetRS232DataBits:
1442     lds     temp0,InputBufferBegin+4 ;first parameter - data bits 0=5db, 1=6db, 2=7db, 3=8db
1443     cpi     temp0,DataBits8         ;if to set 8-bits communication
1444     breq    Databits8or9Set         ;then don't change 8/9 bit communication
1445     in      temp1,UCSRB             ;otherwise load UCSRB
1446     cbr     temp1,(1<<UCSZ2)        ;clear 9-bit communication
1447     out     UCSRB,temp1             ;and write back
1448 Databits8or9Set:
1449     rcall   RS232DataBitsLocal
1450     rjmp    OneZeroAnswer           ;acknowledge reception with single zero

```

```

1451 RS232DataBitsLocal:
1452     rcall    GetUCSRCtotemp1
1453     bst      temp0,0                ;set the UCSZ0
1454     bld      temp1,UCSZ0
1455     bst      temp0,1                ;set the UCSZ1
1456     bld      temp1,UCSZ1
1457     rcall    Settemp1toUCSRC
1458     ret
1459 GetUCSRCtotemp1:
1460     cli                      ;obtain UCSRC
1461     in        temp1,UBRRH
1462     in        temp1,UCSRC        ;to temp1
1463     sei
1464     nop                      ;for to enable possible interrupt waiting before ret instruction (ret has long duration)
1465     ret
1466 Settemp1toUCSRC:
1467     sbr      temp1,(1<<URSEL)    ;will be writing to UCSRC
1468     out       UCSRC,temp1        ;and write back to register with new UCSZ0 and UCSZ1
1469     ret
1470 ;-----
1471 DoGetRS232DataBits:
1472     rcall    GetUCSRCtotemp1
1473     clr      temp0                ;clear answer
1474     bst      temp1,UCSZ0        ;obtain UCSZ0
1475     bld      temp0,0            ;and save to bit 0
1476     bst      temp1,UCSZ1        ;obtain UCSZ1
1477     bld      temp0,1            ;and save to bit 1
1478     mov      R0,temp0           ;return number of databits in R0
1479     rjmp     DoGetIn            ;and finish
1480 ;-----
1481 DoSetRS232Parity:
1482     lds      temp0,InputBufferBegin+4 ;first parameter - parity: 0=none, 1=odd, 2=even, 3=mark, 4=space
1483     cpi      temp0,3
1484     brcc     StableParity
1485     rcall    GetUCSRCtotemp1
1486     cbr      temp1,(1<<UPM1)|(1<<UPM0) ;clear parity bits
1487     cpi      temp0,ParityNone      ;if none
1488     breq     SetParityOut
1489     sbr      temp1,(1<<UPM1)
1490     cpi      temp0,ParityEven      ;if even
1491     breq     SetParityOut
1492     sbr      temp1,(1<<UPM0)
1493     cpi      temp0,ParityOdd       ;if odd
1494     brne     ParityErrorAnswer
1495 SetParityOut:
1496     rcall    Settemp1toUCSRC
1497     in        temp1,UCSRB          ;load UCSRB
1498     cbr      temp1,(1<<UCSZ2)      ;if is 9-bits communication then change it under 9 bits
1499     out       UCSRB,temp1         ;and write back
1500     rjmp     OneZeroAnswer        ;acknowledge reception with single zero
1501 StableParity:
1502     in        temp1,UCSRB          ;change transmitting parity bit TXB8
1503     bst      temp0,0              ;load lowest bit
1504     bld      temp1,TXB8           ;and save to its place TXB8
1505     sbr      temp1,(1<<UCSZ2)      ;set the UCSZ2 bit - 9 bits communication
1506     out       UCSRB,temp1         ;changed TXB8 and UCSZ2 write to UCSRB
1507
1508     ldi      temp0,3              ;set the 9-databit

```

```

1509      rcall    RS232DataBitsLocal      ;and return in temp1 contents UCSRC
1510      cbr      temp1,(1<<UPM1)|(1<<UPM0) ;disable parity
1511      rcall    SettempltoUCSRC
1512      rjmp     OneZeroAnswer           ;acknowledge reception with single zero
1513 ParityErrorAnswer:
1514      rjmp     TwoZeroAnswer           ;acknowledge reception with two zero
1515 ;-----
1516 DoGetRS232Parity:
1517      in        temp1,UCSRB             ;load UCSRB
1518      sbrc      temp1,UCSZ2             ;if is 9-bits communication
1519      rjmp     ParityIsStable          ;then parity is space or mark
1520
1521      rcall    GetUCSRCtotemp1
1522      cbr      temp1,~((1<<UPM0)|(1<<UPM1)) ;and let nonzero only parity bits
1523
1524      cpi      temp1,(1<<UPM0)|(1<<UPM1) ;if are both set
1525      ldi      temp0,ParityOdd          ;this is odd parity
1526      breq     RetGetParity             ;and finish
1527      cpi      temp1,(1<<UPM1)         ;if is UPM1 set
1528      ldi      temp0,ParityEven        ;this is even parity
1529      breq     RetGetParity             ;and finish
1530      ldi      temp0,ParityNone        ;otherwise is that none parity
1531      rjmp     RetGetParity             ;and finish
1532 ParityIsStable:
1533      bst      temp1,TKB8               ;obtain what is 9-th bit
1534      ldi      temp0,ParityMark        ;prepare mark answer
1535      brts     RetGetParity             ;if is 1 then return mark
1536      ldi      temp0,ParitySpace       ;otherwise return space
1537 RetGetParity:
1538      mov      R0,temp0                ;answer move from temp0 to R0
1539      rjmp     DoGetIn                 ;and finish
1540 ;-----
1541 DoSetRS232StopBits:
1542      lds      temp0,InputBufferBegin+4 ;first parameter - stop bit 0=1stopbit 1=2stopbits
1543      rcall    GetUCSRCtotemp1
1544      bst      temp0,0                 ;and lowest bit from parameter
1545      bld      temp1,USBS              ;save as stopbit
1546      rcall    SettempltoUCSRC
1547      rjmp     OneZeroAnswer           ;acknowledge reception with single zero
1548 ;-----
1549 DoGetRS232StopBits:
1550      rcall    GetUCSRCtotemp1
1551      clr      R0                      ;clear answer
1552      bst      temp1,USBS              ;and bit USBS
1553      bld      R0,0                   ;write to answer
1554      rjmp     DoGetIn                 ;and finish
1555 ;-----
1556 ;----- END USER FUNCTIONS ----- END USER FUNCTIONS -----
1557
1558 OneZeroAnswer:      ;send single zero
1559      ldi      temp0,1                 ;number of my bytes answers to temp0
1560      rjmp     ComposeGET_STATUS2
1561 ;----- STANDARD USB REQUESTS ----- STANDARD USB REQUESTS -----
1562 StandardRequest:
1563      cpi      temp1,GET_STATUS        ;
1564      breq     ComposeGET_STATUS      ;
1565
1566      cpi      temp1,CLEAR_FEATURE    ;

```

```

1567      breq    ComposeCLEAR_FEATURE      ;
1568
1569      cpi     temp1,SET_FEATURE          ;
1570      breq    ComposeSET_FEATURE         ;
1571
1572      cpi     temp1,SET_ADDRESS          ;if to set address
1573      breq    ComposeSET_ADDRESS         ;set the address
1574
1575      cpi     temp1,GET_DESCRIPTOR       ;if requested descriptor
1576      breq    ComposeGET_DESCRIPTOR      ;generate it
1577
1578      cpi     temp1,SET_DESCRIPTOR       ;
1579      breq    ComposeSET_DESCRIPTOR      ;
1580
1581      cpi     temp1,GET_CONFIGURATION    ;
1582      breq    ComposeGET_CONFIGURATION   ;
1583
1584      cpi     temp1,SET_CONFIGURATION    ;
1585      breq    ComposeSET_CONFIGURATION   ;
1586
1587      cpi     temp1,GET_INTERFACE        ;
1588      breq    ComposeGET_INTERFACE       ;
1589
1590      cpi     temp1,SET_INTERFACE        ;
1591      breq    ComposeSET_INTERFACE       ;
1592
1593      cpi     temp1,SYNCH_FRAME          ;
1594      breq    ComposeSYNCH_FRAME         ;
1595      ;if not found known request
1596      rjmp    ZeroDATA1Answer            ;if that was something unknown, then prepare zero answer
1597
1598 ComposeSET_ADDRESS:
1599      lds     temp1,InputBufferBegin+4   ;new address to temp1
1600      rcall   SetMyNewUSBAddresses       ;and compute NRZI and bitstuffing coded addresses
1601      ldi     State,AddressChangeState   ;set state for Address changing
1602      rjmp    ZeroDATA1Answer            ;send zero answer
1603
1604 ComposeSET_CONFIGURATION:
1605      lds     temp0,InputBufferBegin+4   ;number of configuration to variable ConfigByte
1606      sts     ConfigByte,temp0           ;
1607 ComposeCLEAR_FEATURE:
1608 ComposeSET_FEATURE:
1609 ComposeSET_INTERFACE:
1610 ZeroStringAnswer:
1611      rjmp    ZeroDATA1Answer            ;send zero answer
1612 ComposeGET_STATUS:
1613 TwoZeroAnswer:
1614      ldi     temp0,2                    ;number of my bytes answers to temp0
1615 ComposeGET_STATUS2:
1616      ldi     ZH, high(StatusAnswer<<1) ;ROMpointer to answer
1617      ldi     ZL, low(StatusAnswer<<1)
1618      rjmp    ComposeEndXXXDescriptor    ;and complete
1619 ComposeGET_CONFIGURATION:
1620      lds     temp0,ConfigByte
1621      and     temp0,temp0                ;if I am unconfigured
1622      breq    OneZeroAnswer              ;then send single zero - otherwise send my configuration
1623      ldi     temp0,1                    ;number of my bytes answers to temp0
1624      ldi     ZH, high(ConfigAnswerMinus1<<1) ;ROMpointer to answer

```

```

1625         ldi     ZL, low(ConfigAnswerMinus1<<1)+1
1626         rjmp    ComposeEndXXXDescriptor      ;and complete
1627 ComposeGET_INTERFACE:
1628         ldi     ZH, high(InterfaceAnswer<<1) ;ROMpointer to answer
1629         ldi     ZL, low(InterfaceAnswer<<1)
1630         ldi     temp0,1                      ;number of my bytes answers to temp0
1631         rjmp    ComposeEndXXXDescriptor      ;and complete
1632 ComposeSYNCH_FRAME:
1633 ComposeSET_DESCRIPTOR:
1634         rcall   ComposeSTALL
1635         ret
1636 ComposeGET_DESCRIPTOR:
1637         lds     temp1,InputBufferBegin+5      ;DescriptorType to temp1
1638         cpi     temp1,DEVICE                  ;DeviceDescriptor
1639         breq    ComposeDeviceDescriptor      ;
1640         cpi     temp1,CONFIGURATION           ;ConfigurationDescriptor
1641         breq    ComposeConfigDescriptor      ;
1642         cpi     temp1,STRING                  ;StringDeviceDescriptor
1643         breq    ComposeStringDescriptor      ;
1644         ret
1645 ComposeDeviceDescriptor:
1646         ldi     ZH, high(DeviceDescriptor<<1) ;ROMpointer to descriptor
1647         ldi     ZL, low(DeviceDescriptor<<1)
1648         ldi     temp0,0x12                    ;number of my bytes answers to temp0
1649         rjmp    ComposeEndXXXDescriptor      ;and complete
1650 ComposeConfigDescriptor:
1651         ldi     ZH, high(ConfigDescriptor<<1) ;ROMpointer to descriptor
1652         ldi     ZL, low(ConfigDescriptor<<1)
1653         ldi     temp0,9+9+7                    ;number of my bytes answers to temp0
1654 ComposeEndXXXDescriptor:
1655         lds     TotalBytesToSend,InputBufferBegin+8 ;number of requested bytes to TotalBytesToSend
1656         cp      TotalBytesToSend,temp0          ;if not requested more than I can send
1657         brcs    HostConfigLength                ;transmit the requested number
1658         mov     TotalBytesToSend,temp0          ;otherwise send number of my answers
1659 HostConfigLength:
1660         mov     temp0,TotalBytesToSend          ;
1661         clr     TransmitPart                    ;zero the number of 8 bytes answers
1662         andi    temp0,0b00000111                ;if is length divisible by 8
1663         breq    Length8Multiply                ;then not count one answer (under 8 byte)
1664         inc     TransmitPart                    ;otherwise count it
1665 Length8Multiply:
1666         mov     temp0,TotalBytesToSend          ;
1667         lsr     temp0                          ;length of 8 bytes answers will reach
1668         lsr     temp0                          ;integer division by 8
1669         lsr     temp0
1670         add     TransmitPart,temp0              ;and by addition to last non entire 8-bytes to variable TransmitPart
1671         ldi     temp0,DATA0PID                  ;DATA0 PID - in the next will be toggled to DATA1PID in load descriptor
1672         sts     OutputBufferBegin+1,temp0        ;store to output buffer
1673         rjmp    ComposeNextAnswerPart
1674 ComposeStringDescriptor:
1675         ldi     temp1,4+8                      ;if RAMread=4(insert zeros from ROM reading) + 8(behind first byte no load zero)
1676         mov     RAMread,temp1
1677         lds     temp1,InputBufferBegin+4        ;DescriptorIndex to temp1
1678         cpi     temp1,0                        ;LANGID String
1679         breq    ComposeLangIDString            ;
1680         cpi     temp1,2                        ;DevNameString
1681         breq    ComposeDevNameString           ;
1682         brcc    ZeroStringAnswer              ;if is DescriptorIndex higher than 2 - send zero answer

```

```

1683                                     ;otherwise is VendorString
1684 ComposeVendorString:
1685     ldi    ZH, high(VendorStringDescriptor<<1)    ;ROMpointer to descriptor
1686     ldi    ZL, low(VendorStringDescriptor<<1)
1687     ldi    temp0, (VendorStringDescriptorEnd-VendorStringDescriptor)*4-2    ;number of my bytes answers to temp0
1688     rjmp   ComposeEndXXXDescriptor    ;and complete
1689 ComposeDevNameString:
1690     ldi    ZH, high(DevNameStringDescriptor<<1)    ;ROMpointer to descriptor
1691     ldi    ZL, low(DevNameStringDescriptor<<1)
1692     ldi    temp0, (DevNameStringDescriptorEnd-DevNameStringDescriptor)*4-2    ;number of my bytes answers to temp0
1693     rjmp   ComposeEndXXXDescriptor    ;and complete
1694 ComposeLangIDString:
1695     clr    RAMread
1696     ldi    ZH, high(LangIDStringDescriptor<<1)    ;ROMpointer to descriptor
1697     ldi    ZL, low(LangIDStringDescriptor<<1)
1698     ldi    temp0, (LangIDStringDescriptorEnd-LangIDStringDescriptor)*2;number of my bytes answers to temp0
1699     rjmp   ComposeEndXXXDescriptor    ;and complete
1700 ;-----
1701 ZeroDATA1Answer:
1702     rcall  ComposeZeroDATA1PIDAnswer
1703     ret
1704 ;-----
1705 SetMyNewUSBAddresses:    ;set new USB addresses in NRZI coded
1706     mov    temp2,temp1    ;address to temp2 and temp1 and temp3
1707     mov    temp3,temp1    ;
1708     cpi    temp1,0b01111111    ;if address contains less than 6 ones
1709     brne   NewAddressNo6ones    ;then don't add bitstuffing
1710     ldi    temp1,0b10111111    ;else insert one zero - bitstuffing
1711 NewAddressNo6ones:
1712     andi   temp3,0b00000111    ;mask 3 low bits of Address
1713     cpi    temp3,0b00000111    ;and if 3 low bits of Address is no all ones
1714     brne   NewAddressNo3ones    ;then no change address
1715     ;else insert zero after 3-rd bit (bitstuffing)
1716     sec
1717     rol    temp2    ;rotate left
1718     andi   temp2,0b11110111    ;and inserted zero after 3-rd bit
1719 NewAddressNo3ones:
1720     sts    MyOutAddressSRAM,temp2    ;store new non-coded address Out (temp2)
1721     ;and now perform NRZI coding
1722     rcall  NRZIforAddress    ;NRZI for AddressIn (in temp1)
1723     sts    MyInAddressSRAM,ACC    ;store NRZI coded AddressIn
1724
1725     lds    temp1,MyOutAddressSRAM    ;load non-coded address Out (in temp1)
1726     rcall  NRZIforAddress    ;NRZI for AddressOut
1727     sts    MyOutAddressSRAM,ACC    ;store NRZI coded AddressOut
1728
1729     ret    ;and return
1730 ;-----
1731 NRZIforAddress:
1732     clr    ACC    ;original answer state - of my nNRZI USB address
1733     ldi    temp2,0b00000001    ;mask for xoring
1734     ldi    temp3,8    ;bits counter
1735 SetMyNewUSBAddressesLoop:
1736     mov    temp0,ACC    ;remember final answer
1737     ror    temp1    ;to carry transmitting bit LSB (in direction firstly LSB then MSB)
1738     brcs   NoXORBits    ;if one - don't change state
1739     eor    temp0,temp2    ;otherwise state will be changed according to last bit of answer
1740 NoXORBits:

```

```

1741         ror     temp0                ;last bit of changed answer to carry
1742         rol     ACC                  ;and from carry to final answer to the LSB place (and reverse LSB and MSB order)
1743         dec     temp3                ;decrement bits counter
1744         brne    SetMyNewUSBAddressesLoop ;if bits counter isn't zero repeat transmitting with next bit
1745         ret
1746 ;-----
1747 ;----- END DATA ENCRYPTION USB REQUESTS -----
1748
1749 PrepareOutContinuousBuffer:
1750         rcall   PrepareContinuousBuffer
1751         rcall   MakeOutBitStuff
1752         ret
1753 ;-----
1754 PrepareContinuousBuffer:
1755         mov     temp0,TransmitPart
1756         cpi     temp0,1
1757         brne    NextAnswerInBuffer    ;if buffer empty
1758         rcall   ComposeZeroAnswer     ;prepare zero answer
1759         ret
1760 NextAnswerInBuffer:
1761         dec     TransmitPart           ;decrement general length of answer
1762 ComposeNextAnswerPart:
1763         mov     temp1,TotalBytesToSend ;decrement number of bytes to transmit
1764         subi    temp1,8                ;is is necessary to send more as 8 byte
1765         ldi     temp3,8                ;if yes - send only 8 byte
1766         brcc    Nad8Bytov
1767         mov     temp3,TotalBytesToSend ;otherwise send only given number of bytes
1768         clr     TransmitPart
1769         inc     TransmitPart           ;and this will be last answer
1770 Nad8Bytov:
1771         mov     TotalBytesToSend,temp1 ;decremented number of bytes to TotalBytesToSend
1772         rcall   LoadXXXDescriptor
1773         ldi     ByteCount,2            ;length of output buffer (only SOP and PID)
1774         add     ByteCount,temp3        ;+ number of bytes
1775         rcall   AddCRCOut              ;addition of CRC to buffer
1776         inc     ByteCount              ;length of output buffer + CRC16
1777         inc     ByteCount
1778         ret                             ;finish
1779 ;-----
1780 .equ    USBversion      =0x0101        ;for what version USB is that (1.01)
1781 .equ    VendorUSBID     =0x03EB        ; vendor identifier (Atmel=0x03EB)
1782 .equ    DeviceUSBID     =0x21FF        ;product identifier (USB to RS232 converter ATmega8=0x21FF)
1783 .equ    DeviceVersion   =0x0003        ;version number of product (version=0.03)
1784                                     ;(0.01=AT90S2313 Infra buffer)
1785                                     ;(0.02=AT90S2313 RS232 buffer 32bytes)
1786                                     ;(0.03=ATmega8 RS232 buffer 800bytes)
1787 .equ    MaxUSBCurrent   =50            ;current consumption from USB (50mA) - together with MAX232
1788 ;-----
1789 DeviceDescriptor:
1790         .db     0x12,0x01              ;0 byte - size of descriptor in byte
1791                                     ;1 byte - descriptor type: Device descriptor
1792         .dw     USBversion              ;2,3 byte - version USB LSB (1.00)
1793         .db     0x00,0x00              ;4 byte - device class
1794                                     ;5 byte - subclass
1795         .db     0x00,0x08              ;6 byte - protocol code
1796                                     ;7 byte - FIFO size in bytes
1797         .dw     VendorUSBID             ;8,9 byte - vendor identifier (Cypress=0x04B4)
1798         .dw     DeviceUSBID            ;10,11 byte - product identifier (teplomer=0x0002)

```



```

1799         .dw      DeviceVersion          ;12,13 byte - product version number (verzia=0.01)
1800         .db      0x01,0x02              ;14 byte - index of string "vendor"
1801                                         ;15 byte - index of string "product"
1802         .db      0x00,0x01              ;16 byte - index of string "serial number"
1803                                         ;17 byte - number of possible configurations
1804 DeviceDescriptorEnd:
1805 ;-----
1806 ConfigDescriptor:
1807         .db      0x9,0x02                ;length, descriptor type
1808 ConfigDescriptorLength:
1809         .dw      9+9+7                    ;entire length of all descriptors
1810 ConfigAnswerMinus1:
1811                                         ;for sending the number - configuration number (attention - addition of 1 required)
1812         .db      1,1                      ;numInterfaces, configuration number
1813         .db      0,0x80                    ;string index, attributes; bus powered
1814         .db      MaxUSBCurrent/2,0x09      ;current consumption, interface descriptor length
1815         .db      0x04,0                    ;interface descriptor; number of interface
1816 InterfaceAnswer:
1817         .db      0,1                      ;for sending number of alternatively interface
1818         .db      0,1                      ;alternatively interface; number of endpoints except EP0
1819 StatusAnswer:
1820         .db      0,0                      ;2 zero answers (saving ROM place)
1821         .db      0,0                      ;interface class; interface subclass
1822         .db      0,0                      ;protocol code; string index
1823         .db      0x07,0x5                  ;length, descriptor type - endpoint
1824         .db      0x81,0                    ;endpoint address; transfer type
1825         .dw      0x08                      ;max packet size
1826         .db      10,0                     ;polling interval [ms]; dummy byte (for filling)
1827 ConfigDescriptorEnd:
1828 ;-----
1829 LangIDStringDescriptor:
1830         .db      (LangIDStringDescriptorEnd-LangIDStringDescriptor)*2,3 ;length, type: string descriptor
1831         .dw      0x0409                    ;English
1832 LangIDStringDescriptorEnd:
1833 ;-----
1834 VendorStringDescriptor:
1835         .db      (VendorStringDescriptorEnd-VendorStringDescriptor)*4-2,3 ;length, type: string descriptor
1836 Copyright:
1837         .db      "Ing. Igor Cesko http://www.cesko.host.sk"
1838 CopyrightEnd:
1839 VendorStringDescriptorEnd:
1840 ;-----
1841 DevNameStringDescriptor:
1842         .db      (DevNameStringDescriptorEnd-DevNameStringDescriptor)*4-2,3;length, type: string descriptor
1843         .db      "AVR309: USB to UART protocol converter"
1844 DevNameStringDescriptorEnd:
1845 ;-----
1846 ;*****
1847 ;* End of program
1848 ;*****
1849 ;-----
1850 ;* End of file
1851 ;*****

```