

MACRO ASSEMBLER A51 V6.10
 OBJECT MODULE PLACED IN .\BLOCKIO.OBJ
 ASSEMBLER INVOKED BY: C:\Keil\C51\BIN\A51.EXE .\BLOCKIO.A51 REGISTERBANK(0) SET(SMALL) DEBUG EP

```

LOC OBJ          LINE    SOURCE
1          NAME      HIDwithBlockIO
2          ; This example adds a BlockIO function to the basic HID example
3          ;
4          ; Copyright (C) 2001, Intel Corporation
5          ; All rights reserved.
6          ; Permission is hereby granted to merge this program code with other program
7          ; material to create a derivative work. This derivative work may be distributed
8          ; in compiled object form only. Any other publication of this program, in any form,
9          ; without the explicit permission of the copyright holder is prohibited.
10         ;
11         ; Send questions and comments to John.Hyde@intel.com
12         ;
13         ;
14         ; This example adds a BlockIO function to the basic HID example: This involved:
15         ;   a) Adding an Interface Descriptor for BlockIO
16         ;   b) Adding two endpoints to support bulk read and write
17         ;   c) Adding the code to support endpoint enable/read/write/disable
18         ;   d) Several String Descriptors were also added
19         ; This example is therefore a Composite Device
20         ;
21         ; This BlockIO example sends and receives bulkio packets
22         ; Up to 3FFFH (16,384) bytes can be transferred
23         ; Commands/status are sent/read using a HID interface
24         ; Data is sent/read using a BlockIO interface
25         ;
26         ; To get the best throughput on BlockIO the bulk endpoints are polled.
27         ; This works because the Main loop is so short and polling saves all of
28         ; the interrupt processing.
29         ; Register save must now be added to the real interrupt routines
30         ;
31         ; Derived from BAL Version 3.0
32         ;
33         ; This version works with dScope monSIO0.hex (uses Serial Port 0, loads at 1200H)
34         ;
35         ; Changes from Version 2.0
36         ;   a) Two misplaced labels corrected
37         ;   b) CommandTable moved outside of the constrained page
38         ;   c) SETUPDAT buffer copied to direct access memory, simplified coding
39         ;   d) Optional Set_Idle now not supported (returns STALL, not ignorred)
40         ;   e) R7 used in place of Temp, saved code space
41         ;   f) Code reorganized to separate hardware dependant sections
42         ;
43         ;           Old           New
44         ;           USBINT.A51    Decode.A51
45         ;           Vectors.A51   EZInt.A51
46         ;           Timer.A51     EZInt.A51
47         ;           Main.A51      EZMain.A51
48         ;   g) EP0Size made an equate to ease coding of other components
0040      EP0Size EQU    64          ; For EZ-USB
49         ;   h) Code added for descriptors > EP0Size
50         ;
51         ; Changes from Version 1.0
52         ;   a) Register saving removed from Vectors.A51
53         ;           Main has no context which needs to be saved
54         ;   b) There was a race condition in USB_INT::Set_Report: which
55         ;           could cause OLD data to be read. Busy algorithm changed
56         ;   c) The USB Version# was incorrectly declared in the Device Descriptor
57         ;           It was 0101H and is now 0110H (data from USB IF)
58         ;

```

```

59      ;$include (Declare.A51)
+1     60      ; This module declares the variables and constants used in the examples
+1     61      ; It is common to all of the examples
+1     62      ;
+1     63      ; Declare Special Function Registers used
0088   +1     64      TimerControl    DATA    088H
0089   +1     65      TimerMode      DATA    089H
008C   +1     66      Timer0HIGH     DATA    08CH
008A   +1     67      Timer0LOW      DATA    08AH
008C   +1     68      Timer0_Enable  EQU     TimerControl.4
00A8   +1     69      EI              DATA    0A8H
00E8   +1     70      EIE             DATA    0E8H      ; EZ-USB specific
0091   +1     71      EXIF           DATA    091H      ; EZ-USB specific
00D8   +1     72      EICON          DATA    0D8H      ; EZ-USB specific
0092   +1     73      PageReg        DATA    092H      ; EZ-USB specific, used with MOVX @Ri
0086   +1     74      DPS            DATA    086H      ; EZ-USB specific, used with dual data pointers
0084   +1     75      DPL1           DATA    084H
0085   +1     76      DPH1           DATA    085H
008E   +1     77      ClockControl   DATA    08EH      ; EZ-USB specific
+1     78      ;
+1     79      ; "External" memory locations used, EZ-USB specific
+1     80      ; Note that most of these variables are in Page 7FH
7FE8   +1     81      SETUPDAT       EQU     07FE8H
7FD4   +1     82      SUDPTR         EQU     07FD4H
7FE3   +1     83      AutoPtrHIGH    EQU     07FE3H
7FE4   +1     84      AutoPtrLOW     EQU     07FE4H
7FE5   +1     85      AutoPtrData    EQU     07FE5H
7FDD   +1     86      EndpointPairing EQU     07FDDH
7FB4   +1     87      EP0Control     EQU     07FB4H
7F00   +1     88      EP0InBuffer    EQU     07F00H
7EC0   +1     89      EP0OutBuffer   EQU     07EC0H      ; Not in Page 7FH
7FB5   +1     90      EP0InByteCount EQU     07FB5H
7FC5   +1     91      EP0OutByteCount EQU     07FC5H
7E80   +1     92      EP1InBuffer    EQU     07E80H      ; Not in Page 7FH
7E40   +1     93      EP1OutBuffer   EQU     07E40H      ; Not in Page 7FH
7FB7   +1     94      EP1InByteCount EQU     07FB7H
7FC7   +1     95      EP1OutByteCount EQU     07FC7H
7FB9   +1     96      EP2InByteCount EQU     07FB9H
7FC9   +1     97      EP2OutByteCount EQU     07FC9H
7E00   +1     98      EP2InBuffer    EQU     07E00H      ; Not in Page 7FH
7DC0   +1     99      EP2OutBuffer   EQU     07DC0H      ; Not in Page 7FH
7FB8   +1    100      EP2InStatus     EQU     07FB8H
7FC8   +1    101      EP2OutStatus    EQU     07FC8H
7FAC   +1    102      IN07IEN        EQU     07FACH
7FA9   +1    103      IN07IRQ        EQU     07FA9H
7FAD   +1    104      OUT07IEN       EQU     07FADH
7FAA   +1    105      OUT07IRQ       EQU     07FAAH
7FAE   +1    106      USBIEN         EQU     07FAEH
7FAB   +1    107      USBIRQ        EQU     07FABH
7FD6   +1    108      USBControl     EQU     07FD6H
7FA6   +1    109      I2CData        EQU     07FA6H
7FA5   +1    110      I2CControl     EQU     07FA5H
7F93   +1    111      PortA_Config    EQU     07F93H
7F94   +1    112      PortB_Config    EQU     07F94H
7F95   +1    113      PortC_Config    EQU     07F95H
7F96   +1    114      PortA_OUT      EQU     07F96H
7F97   +1    115      PortB_OUT      EQU     07F97H
7F98   +1    116      PortC_OUT      EQU     07F98H
7F99   +1    117      PortA_PINS     EQU     07F99H
7F9A   +1    118      PortB_PINS     EQU     07F9AH
7F9B   +1    119      PortC_PINS     EQU     07F9BH
7F9C   +1    120      PortA_OE       EQU     07F9CH
7F9D   +1    121      PortB_OE       EQU     07F9DH
7F9E   +1    122      PortC_OE       EQU     07F9EH
+1    123
8000   +1    124      DataBuffer     EQU     08000H      ; Used to store the Bulk IO packets

```

```

+1 125 ;
+1 126 ; Byte Variables
+1 127
----
+1 128 DSEG AT 20H
0020 +1 129 FLAGS: DS 1 ; This register is bit-addressable
+1 130 ; Bit Variables
0000 +1 131 Configured EQU FLAGS.0 ; Is this device configured
0001 +1 132 STALL EQU FLAGS.1 ; Need to STALL endpoint 0
0002 +1 133 SendData EQU FLAGS.2 ; Need to send data to PC Host
0003 +1 134 IsDescriptor EQU FLAGS.3 ; Enable a shortcut reply
0004 +1 135 SetAddress EQU FLAGS.4 ; Set the SIE address
+1 136 ;
0021 +1 137 MonitorSpace: DS 1FH ; Used by Dscope
+1 138 ;Expired_Time: DS 1 ; A downcounter for timed Reports
0040 +1 139 ReplyCount: DS 1 ; Byte count for following buffer
0041 +1 140 ReplyBuffer: DS 2 ; Buffer for immediate reply
0043 +1 141 CurrentConfiguration:
0043 +1 142 DS 1 ; Some examples support > 1 configurations
0044 +1 143 SaveACC: DS 1 ; Used in interrupt service routine
0045 +1 144 SavePSW: DS 1 ; Used in interrupt service routine
0046 +1 145 SaveDPH: DS 1 ; Needed to save Descriptor Pointer ..
0047 +1 146 SaveDPL: DS 1 ; .. for descriptors > EP0Size
0048 +1 147 SaveLength: DS 1 ; Number of bytes still to send
0049 +1 148 SetupData: ; Buffer in direct access memory
0049 +1 149 RequestType: DS 1
004A +1 150 Request: DS 1
004B +1 151 wValueLow: DS 1
004C +1 152 wValueHigh: DS 1
004D +1 153 wIndexLow: DS 1
004E +1 154 wIndexHigh: DS 1
004F +1 155 wLengthLow: DS 1
0050 +1 156 wLengthHigh: DS 1
+1 157 ;
0051 +1 158 Old_Buttons: DS 1
0052 +1 159 LEDstrobe: DS 1
0053 +1 160 LEDvalue: DS 1
0054 +1 161 Msec_Counter: DS 1
+1 162 ;
0055 +1 163 PacketCounter: DS 1
0056 +1 164 LastPacketValidBytes:DS 1
+1 165
+1 166
+1 167 ;$include (EZInt.A51)
+1 168 ; This module contains all the EZUSB-specific hardware code
+1 169 ; This module also contains all of the interrupt vector declarations and
+1 170 ; the first level interrupt servicing (register save, call subroutine,
+1 171 ; clear interrupt source, restore registers, return)
+1 172 ; Suspend and Resume are handled totally in this module
+1 173 ;
+1 174 ; A Reset sends us to Program space location 0
----
+1 175 CSEG AT 0 ; Code space
+1 176 USING 0 ; Reset forces Register Bank 0
0000 02130F +1 177 LJMP Reset
+1 178 ;
+1 179 ; The interrupt vector table is also located here
+1 180 ; EZ-USB has two levels of USB interrupts:
+1 181 ; 1-the main level is described in this table (at ORG 43H)
+1 182 ; 2-there are 21 sources of USB interrupts and these are described in USB_ISR
+1 183 ; This means that two levels of acknowledgement and clearing will be required
+1 184 ; LJMP INT0_ISR ; Features not used are commented out
+1 185 ; ORG 0BH
+1 186 ; LJMP Timer0_ISR
+1 187 ; ORG 13H
+1 188 ; LJMP INT1_ISR
+1 189 ; ORG 1BH
+1 190 ; LJMP Timer1_ISR

```

```
+1 191 ; ORG 23H
+1 192 ; LJMP UART0_ISR
+1 193 ; ORG 2BH
+1 194 ; LJMP Timer2_ISR
+1 195 ; ORG 33H
+1 196 ; LJMP WakeUp_ISR
+1 197 ; ORG 3BH
+1 198 ; LJMP UART1_ISR
0043 +1 199 ORG 43H
0043 021200 +1 200 LJMP USB_ISR ; Auto Vector will replace byte 45H
+1 201 ; ORG 4BH
+1 202 ; LJMP I2C_ISR
+1 203 ; ORG 53H
+1 204 ; LJMP INT4_ISR
+1 205 ; ORG 5BH
+1 206 ; LJMP INT5_ISR
+1 207 ; ORG 63H
+1 208 ; LJMP INT6_ISR
+1 209
1200 +1 210 ORG 1200H ; Load above monSIO0.hex
1200 021289 +1 211 USB_ISR:LJMP SUDAV_ISR
1203 00 +1 212 DB 0 ; Pad entries to 4 bytes
1204 021270 +1 213 LJMP SOF_ISR
1207 00 +1 214 DB 0
1208 021227 +1 215 LJMP SUTOK_ISR
120B 00 +1 216 DB 0
120C 021237 +1 217 LJMP Suspend_ISR
120F 00 +1 218 DB 0
1210 021232 +1 219 LJMP USBReset_ISR
1213 00 +1 220 DB 0
1214 021227 +1 221 LJMP Reserved
1217 00 +1 222 DB 0
1218 021250 +1 223 LJMP EP0In_ISR
121B 00 +1 224 DB 0
121C 021227 +1 225 LJMP EP0Out_ISR
121F 00 +1 226 DB 0
1220 021227 +1 227 LJMP EP1In_ISR
1223 00 +1 228 DB 0
1224 021227 +1 229 LJMP EP1Out_ISR
+1 230 ; End of Interrupt Vector tables
+1 231
+1 232 ; When a feature is used insert the required interrupt processing here
+1 233 ; The example use only used Endpoints 0 and 1 and also SOF for timing
1227 +1 234 Reserved:
1227 +1 235 SUTOK_ISR:
1227 +1 236 EP0Out_ISR:
1227 +1 237 EP1In_ISR:
1227 +1 238 EP1Out_ISR:
1227 +1 239 Not_Used: ; Should not get any of these
1227 5129 +1 240 CALL HALT
1229 80FE +1 241 HALT: JMP HALT ; Stack will tell us how we got here!
+1 242
122B +1 243 ClearINT2: ; Tell the hardware that we're done
122B E591 +1 244 MOV A, EXIF
122D C2E4 +1 245 CLR ACC.4 ; Clear the Interrupt 2 bit
122F F591 +1 246 MOV EXIF, A
1231 22 +1 247 RET
+1 248
1232 +1 249 USBReset_ISR: ; Bus has been Reset, move to DEFAULT state
1232 B153 +1 250 ACALL Deconfigured
1234 512B +1 251 CALL ClearINT2
+1 252 ; No need to clear source of interrupt
1236 32 +1 253 RETI
+1 254
1237 +1 255 Suspend_ISR: ; SIE detected an Idle bus
1237 E587 +1 256 MOV A, PCON
```

```
1239 4401      +1 257          ORL    A, #1
123B F587      +1 258          MOV    PCON, A          ; Go to sleep!
123D 00        +1 259          NOP
123E 00        +1 260          NOP                    ; Wake up here due to a USBResume
123F 00        +1 261          NOP
1240 512B      +1 262          CALL   ClearINT2
1242 32        +1 263          RETI
              +1 264
1243           +1 265          WakeUp_ISR:                ; Not using external WAKEUP
              +1 266                    ; So this must be due to a USBResume
1243 C2DC      +1 267          CLR    EICON.4          ; Clear the wakeup interrupt source
1245 32        +1 268          RETI
              +1 269
1246           +1 270          SaveRegisters:
              +1 271          ; Register usage
              +1 272          ; Reg:  MAIN:  Interrupt service routines:
              +1 273          ; ----  -----
              +1 274          ; R0-7  RBO    RB1
              +1 275          ; DPTR  DPTR1  DPTR0
1246 F544      +1 276          MOV    SaveACC, A
1248 85D045     +1 277          MOV    SavePSW, PSW
124B D2D3      +1 278          SETB  PSW.3            ; Select Register Bank 1
124D 0586      +1 279          INC   DPS              ; Switch back to DPTR0
124F 22        +1 280          RET
              +1 281
1250           +1 282          EP0In_ISR:                ; A packet has been read by PC host
1250 5146      +1 283          CALL   SaveRegisters
1252 E548      +1 284          MOV    A, SaveLength    ; Do I have any more data to send?
1254 6008      +1 285          JZ    NoMoreToSend
1256 854683    +1 286          MOV    DPH, SaveDPH    ; Retrieve descriptor pointer
1259 854782    +1 287          MOV    DPL, SaveDPL
125C 51D7      +1 288          CALL   SendNextPieceOfDescriptor
125E           +1 289          NoMoreToSend:
125E 9001A9     +1 290          MOV    DPTR, #(100H OR LOW(IN07IRQ))
1261           +1 291          ExitISR:                ; Common exit for all ISR's
              +1 292          ; On entry DPH = Interrupt ID, DPL = LOW(Interrupt Register)
1261 512B      +1 293          CALL   ClearINT2
1263 747F      +1 294          MOV    A, #7FH        ; EZ-USB I/O Register Page
1265 C583      +1 295          XCH   A, DPH
1267 F0        +1 296          MOVX  @DPTR, A          ; Clear source of interrupt
1268 E544      +1 297          MOV    A, SaveACC
126A 8545D0    +1 298          MOV    PSW, SavePSW    ; Also re-selects Register Bank 0
126D 0586      +1 299          INC   DPS              ; Return to DPTR1
126F 32        +1 300          RETI
              +1 301
1270           +1 302          SOF_ISR:                ; A Start-Of-Frame packet has been received
1270 5146      +1 303          CALL   SaveRegisters
              +1 304          ; This routine services the real time interrupt
              +1 305          ; It is also responsible for the "real world" buttons and lights
              +1 306          ;
1272           +1 307          ServiceTimerRoutine:
1272 D5540F     +1 308          DJNZ  Msec_counter, Done    ; Only need to check every 4msec
1275 755404     +1 309          MOV    Msec_counter, #4      ; Reinitialize
              +1 310          ; LED task
1278 E553      +1 311          MOV    A, LEDValue
127A 907F97     +1 312          MOV    DPTR, #PortB_Out
127D F0        +1 313          MOVX  @DPTR, A          ; Update the real world
              +1 314          ;
              +1 315          ; Create an Input Report from the Buttons value
              +1 316          ; This will be continually overwritten while the PCHost is not polling for data
127E           +1 317          ReadButtons:
127E 907F99     +1 318          MOV    DPTR, #PortA_Pins
1281 E0        +1 319          MOVX  A, @DPTR
1282 9187      +1 320          CALL   CreateInputReport
1284 9002AB     +1 321          Done:  MOV    DPTR, #(200H OR LOW(USBIRQ))
1287 80D8      +1 322          JMP    ExitISR
```

```
+1 323
1289 +1 324 SUDAV_ISR: ; A Setup packet has been received
1289 5146 +1 325 CALL SaveRegisters
128B 754800 +1 326 MOV SaveLength, #0 ; Clear any pending transactions (if any)
128E 907FE8 +1 327 MOV DPTR, #SETUPDAT ; Copy packet to direct access memory
1291 7849 +1 328 MOV R0, #SetupData
1293 7F08 +1 329 MOV R7, #8
1295 E0 +1 330 CopySD: MOVX A, @DPTR
1296 F6 +1 331 MOV @R0, A
1297 A3 +1 332 INC DPTR
1298 08 +1 333 INC R0
1299 DFFA +1 334 DJNZ R7, CopySD
129B 9196 +1 335 CALL ServiceSetupPacket ; Handle the decode of the Setup packet
+1 336 ; if SetAddress { Update SIE address } // NOP on EZ-USB
+1 337 ; if STALL { Stall the endpoint }
+1 338 ; if SendData {
+1 339 ;     if IsDescriptor { send DPTR->descriptor, A = length }
+1 340 ;     else { send ReplyBuffer }
+1 341 ; }
129D 200126 +1 342 JB STALL, SendSTALL
12A0 300216 +1 343 JNB SendData, HandShake
12A3 200324 +1 344 JB IsDescriptor, LoadEP0
+1 345 ; Send data in ReplyBuffer
12A6 907F01 +1 346 MOV DPTR, #EP0InBuffer+1
12A9 7842 +1 347 MOV R0, #ReplyBuffer+1
12AB 7F02 +1 348 MOV R7, #2 ; Copy the two byte buffer
12AD E6 +1 349 CopyRB: MOV A, @R0
12AE F0 +1 350 MOVX @DPTR, A
12AF 1582 +1 351 DEC DPL
12B1 18 +1 352 DEC R0
12B2 DFF9 +1 353 DJNZ R7, CopyRB
12B4 E6 +1 354 MOV A, @R0 ; Get BufferCount
12B5 +1 355 SendEP0InBuffer:
12B5 907FB5 +1 356 MOV DPTR, #EP0InByteCount
12B8 +1 357 StartXfer:
12B8 F0 +1 358 MOVX @DPTR, A ; This write initiates the transfer
12B9 +1 359 HandShake: ; Handshake with host
12B9 7F02 +1 360 MOV R7, #00000010b ; Set HSNACK to tell the SIE that we're done
12BB +1 361 SetEP0Control:
12BB 907FB4 +1 362 MOV DPTR, #EP0Control
12BE E0 +1 363 MOVX A, @DPTR
12BF 4F +1 364 ORL A, R7
12C0 F0 +1 365 MOVX @DPTR, A ; We're done
12C1 9001AB +1 366 MOV DPTR, #(100H OR LOW(USBIRQ))
12C4 809B +1 367 JMP ExitISR
12C6 +1 368 SendSTALL: ; Invalid Request was received
12C6 7F03 +1 369 MOV R7, #00000011b ; Set EPOSTALL and HSNACK
12C8 80F1 +1 370 JMP SetEP0Control
12CA +1 371 LoadEP0: ; Send the data pointed to by DPTR
12CA FF +1 372 MOV R7, A ; Save LENGTH
+1 373 ; Need to return the smaller of "Requested Length" and "Actual Length"
+1 374 ; If "Requested Length" > 255 then use "Actual Length"
+1 375 ; There are no descriptors > 255 in this example
12CB E550 +1 376 MOV A, wLengthHigh
12CD 7007 +1 377 JNZ UseActual
12CF C3 +1 378 CLR C
12D0 954F +1 379 SUBB A, wLengthLow
12D2 E54F +1 380 MOV A, wLengthLow ; This does not affect Carry
12D4 5001 +1 381 JNC UsewLengthLow
12D6 +1 382 UseActual:
12D6 EF +1 383 MOV A, R7
12D7 +1 384 UsewLengthLow:
12D7 +1 385 SendNextPieceOfDescriptor: ; DPTR -> Descriptor to be sent
12D7 FF +1 386 MOV R7, A ; Save LENGTH again
12D8 754800 +1 387 MOV SaveLength, #0 ; Default case, overwrite if necessary
+1 388 ; Do I have more than a single packet to send?
```

```
12DB C3      +1 389          CLR      C
12DC 9440    +1 390          SUBB     A, #EP0Size
12DE 4015    +1 391          JC       SendPacket
              +1 392          ; Need to send multiple packets.
              +1 393          ; Calculate and save address of next packet, send next packet now
12E0 F548    +1 394          MOV      SaveLength, A      ; Send these next time
12E2 7F40    +1 395          MOV      R7, #EP0Size
12E4 C083    +1 396          PUSH     DPH                ; Save current pointer
12E6 C082    +1 397          PUSH     DPL
12E8 EF      +1 398          MOV      A, R7              ; Retrieve length
12E9 91D9    +1 399          CALL     BumpDPTR
12EB 858346  +1 400          MOV      SaveDPH, DPH
12EE 858247  +1 401          MOV      SaveDPL, DPL
12F1 D082    +1 402          POP      DPL
12F3 D083    +1 403          POP      DPH
12F5         +1 404          SendPacket:
12F5 EF      +1 405          MOV      A, R7              ; Retrieve length
12F6 FE      +1 406          MOV      R6, A              ; Save length in R6 for move
12F7 7800    +1 407          MOV      R0, #LOW(EP0InBuffer) ; PageReg = 7FH = HIGH(EP0InBuffer)
12F9 E0      +1 408          CopySTD:MOVX A, @DPTR
12FA F2      +1 409          MOVX     @R0, A
12FB A3      +1 410          INC      DPTR
12FC 08      +1 411          INC      R0
12FD DEFA    +1 412          DJNZ    R6, CopySTD
12FF EF      +1 413          MOV      A, R7              ; Retrieve LENGTH
1300 80B3    +1 414          JMP      SendEP0InBuffer
              +1 415
1302         +1 416          GetOutputReport:          ; Wait for this, it's next on USB
1302 907FC5  +1 417          MOV      DPTR, #EP0OutByteCount ; Enable EP0OutBuffer to receive data
1305 F0      +1 418          MOVX     @DPTR, A          ; Any value will do
1306 907FB4  +1 419          MOV      DPTR, #EP0Control     ; Wait for valid data in EP0OutBuffer
1309 E0      +1 420          Wait40:MOVX A, @DPTR
130A 5408    +1 421          ANL     A, #00001000b        ; Check OUTBSY
130C 70FB    +1 422          JNZ     Wait40
130E 22      +1 423          RET
              +1 424
              +1 425
              +1 426
              +1 427
              +1 428          ;$include (EZMain.A51)
              +1 429          ; This module initializes the microcontroller then executes MAIN forever
              +1 430          ; It is hardware dependant
              +1 431
130F         +1 432          Reset:
130F 74FF    +1 433          MOV      A, #0FFH           ; Debug Aid:
1311 F8      +1 434          MOV      R0, A              ; Set all of internal memory to FF
1312         +1 435          ClearMemory:
1312 F6      +1 436          MOV      @R0, A
1313 D8FD    +1 437          DJNZ    R0, ClearMemory    ; This doesn't set 0 (=R0!)
              +1 438
1315 7581DF  +1 439          MOV      SP, #0DFH         ; Initialize the Stack
1318 75927F  +1 440          MOV      PageReg, #7FH     ; Allows MOVX Ri to access EZ-USB memory
131B 758601  +1 441          MOV      DPS, #1           ; Select DPTR1 for MAIN program
              +1 442          ; DPTR0 is used in interrupt routines
              +1 443
131E 78D6    +1 444          MOV      R0, #Low(USBControl) ; Simulate a disconnect
1320 E2      +1 445          MOVX     A, @R0
1321 54F3    +1 446          ANL     A, #11110011b      ; Clear DISCON, DISCOE
1323 F2      +1 447          MOVX     @R0, A
1324 9149    +1 448          CALL     Wait1second       ; Give the host time to react
1326 E2      +1 449          MOVX     A, @R0           ; Reconnect with this new identity
1327 4406    +1 450          ORL     A, #00000110b     ; Set DISCOE to enable pullup resistor
1329 F2      +1 451          MOVX     @R0, A           ; Set RENUM so that 8051 handles USB requests
132A E4      +1 452          CLR     A
132B F520    +1 453          MOV      FLAGS, A         ; Start in Default state
132D         +1 454          TurnOffLEDs:
```

```
132D F553      +1  455          MOV     LEDValue, A
132F F551      +1  456          MOV     Old_Buttons, A
1331 04        +1  457          INC     A                      ; = 1
1332 F552      +1  458          MOV     LEDstrobe, A
1334           +1  459      Initialize4msecCounter:
1334 F554      +1  460          MOV     Msec_counter, A
1336           +1  461      InitializeIOSystem:          ; Setup for Simmbus A=input, B=output
                                           ; C=External RD#,WR#,TD0,TR0
1336 E58E      +1  463          MOV     A, ClockControl
1338 54F8      +1  464          ANL     A, #11111000b        ; Fastest timing for External memory
133A F58E      +1  465          MOV     ClockControl, A
133C 7893      +1  466          MOV     R0, #LOW(PortA_Config) ; PageReg = 7F = HIGH(PortA_Config)
133E 799C      +1  467          MOV     R1, #LOW(PortA_OE)
1340 E4        +1  468          CLR     A
1341 F2        +1  469          MOVX   @R0, A                ; No alternate functions
1342 F3        +1  470          MOVX   @R1, A                ; Enable PortA for Input
1343 08        +1  471          INC     R0                    ; Point to PortB_Config
1344 09        +1  472          INC     R1                    ; Point to PortB_OE
1345 F2        +1  473          MOVX   @R0, A                ; No alternate functions
1346 F4        +1  474          CPL     A                      ; = 0FFH
1347 F3        +1  475          MOVX   @R1, A                ; Enable PortB for Output
1348 08        +1  476          INC     R0                    ; Point to PortC_Config
1349 09        +1  477          INC     R1                    ; Point to PortC_OE
134A 74C3      +1  478          MOV     A, #11000011b
134C F2        +1  479          MOVX   @R0, A                ; Alternate functions on [7,6,1,0]
134D 74C2      +1  480          MOV     A, #11000010b
134F F3        +1  481          MOVX   @R1, A                ; Most alternate functions are outputs
1350           +1  482      InitializeEndpoints:
1350 78DD      +1  483          MOV     R0, #LOW(EndpointPairing)
1352 7409      +1  484          MOV     A, #00001001b        ; Pair EP2Out/EP3Out and EP2In/EP3In
1354 F2        +1  485          MOVX   @R0, A
1355           +1  486      InitializeTimers:
1355 E589      +1  487          MOV     A, TimerMode          ; Get current mode
1357 54F0      +1  488          ANL     A, #11110000b        ; Clear Timer0 values
1359 04        +1  489          INC     A                      ; Set Timer0 to mode 1 = 16bit counter
135A F589      +1  490          MOV     TimerMode, A
135C C28C      +1  491          CLR     Timer0_Enable        ; Turn off timer 0
135E           +1  492      InitializeInterruptSystem:      ; First initialize the USB level
135E 7401      +1  493          MOV     A, #00000001b
1360 78AC      +1  494          MOV     R0, #LOW(IN07IEN)
1362 F2        +1  495          MOVX   @R0, A                ; Enable interrupts from EP0IN only
1363 08        +1  496          INC     R0
1364 E4        +1  497          CLR     A
1365 F2        +1  498          MOVX   @R0, A                ; Disable interrupts from OUT Endpoints 0-7
1366 08        +1  499          INC     R0
1367 7403      +1  500          MOV     A, #00000011b
1369 F2        +1  501          MOVX   @R0, A                ; Enable (Resume, Suspend,) SOF and SUDAV INTs
136A 08        +1  502          INC     R0
136B 7401      +1  503          MOV     A, #00000001b
136D F2        +1  504          MOVX   @R0, A                ; Enable Auto Vectoring for USB interrupts
                                           ; Now enable the main level
136E 75E801    +1  506          MOV     EIE, #00000001b      ; Enable INT2 = USB Interrupt (only)
1371 75A890    +1  507          MOV     EI, #10010000b      ; Enable interrupt subsystem (and Ser0 for
dScope)
                                           +1  508
                                           +1  509      ; Initialization Complete.
                                           +1  510      ;
1374           +1  511      MAIN:
1374 717D      +1  512          ACALL  DoTaskList        ; BlockIO is polled for performance
1376 80FC      +1  513          JMP     MAIN
                                           +1  514
1378           +1  515      EnableDevice:
1378 22        +1  516          RET
1379           +1  517      DisableDevice:
1379 75F000     +1  518          MOV     B, #0              ; Stop the ECHO function
137C 22        +1  519          RET
                                           +1  520
```

```
137D      +1 521      DoTaskList:
137D E5F0  +1 522          MOV     A, B           ; Get task list (8 tasks)
137F 20E004 +1 523          JB      ACC.0, TASK0      ; Receive BlockIO from PC Host
1382 20E163 +1 524          JB      ACC.1, TASK1      ; Prepare BlockIO for PC Host
1385 22     +1 525          RET
          +1 526
1386      +1 527      TASK0:
          +1 528      ; PC Host is about to send a BlockIO buffer.  Get ready
1386 908000 +1 529          MOV     DPTR, #DataBuffer
1389 E4     +1 530          CLR     A
138A F555   +1 531          MOV     PacketCounter, A
138C F58A   +1 532          MOV     Timer0LOW, A           ; Clear Timer 0
138E F58C   +1 533          MOV     Timer0HIGH, A
1390 79C8   +1 534          MOV     R1, #LOW(EP2OutStatus)
1392      +1 535      WaitForPacket:
1392 E3     +1 536          MOVX    A, @R1
1393 20E1FC +1 537          JB      ACC.1, WaitForPacket ; Wait for Out2BSY = 0
1396 D28C   +1 538          SETB   Timer0_Enable ; Time the data transfer
          +1 539      ; Receive the packet as fast as possible
          +1 540      ; First, set up some loop variables
1398 78E3   +1 541          MOV     R0, #LOW(AutoPtrHIGH) ; Use AutoPtr to speed execution
139A 747D   +1 542          MOV     A, #HIGH(EP2OutBuffer) ; Overflows every time!
139C F2     +1 543          MOVX    @R0, A
139D 08     +1 544          INC     R0           ; Point to AutoPtrLOW
139E 74C0   +1 545          MOV     A, #LOW(EP2OutBuffer)
13A0 F2     +1 546          MOVX    @R0, A
13A1 08     +1 547          INC     R0           ; Point to AutoPtr.Data
13A2 7F04   +1 548          MOV     R7, #4           ; Assume a full buffer (4*16=64)
13A4      +1 549      EP2OutLoop:
13A4 E2     +1 550          MOVX    A, @R0           ; Get new data. This also bumps ptr
13A5 F0     +1 551          MOVX    @DPTR, A        ; Save new data
13A6 A3     +1 552          INC     DPTR          ; Unroll loop to speed execution time
13A7 E2     +1 553          MOVX    A, @R0           ; Unroll loop 2
13A8 F0     +1 554          MOVX    @DPTR, A
13A9 A3     +1 555          INC     DPTR
13AA E2     +1 556          MOVX    A, @R0           ; Unroll loop 3
13AB F0     +1 557          MOVX    @DPTR, A
13AC A3     +1 558          INC     DPTR
13AD E2     +1 559          MOVX    A, @R0           ; Unroll loop 4
13AE F0     +1 560          MOVX    @DPTR, A
13AF A3     +1 561          INC     DPTR
13B0 E2     +1 562          MOVX    A, @R0           ; Unroll loop 5
13B1 F0     +1 563          MOVX    @DPTR, A
13B2 A3     +1 564          INC     DPTR
13B3 E2     +1 565          MOVX    A, @R0           ; Unroll loop 6
13B4 F0     +1 566          MOVX    @DPTR, A
13B5 A3     +1 567          INC     DPTR
13B6 E2     +1 568          MOVX    A, @R0           ; Unroll loop 7
13B7 F0     +1 569          MOVX    @DPTR, A
13B8 A3     +1 570          INC     DPTR
13B9 E2     +1 571          MOVX    A, @R0           ; Unroll loop 8
13BA F0     +1 572          MOVX    @DPTR, A
13BB A3     +1 573          INC     DPTR
13BC E2     +1 574          MOVX    A, @R0           ; Unroll loop 9
13BD F0     +1 575          MOVX    @DPTR, A
13BE A3     +1 576          INC     DPTR
13BF E2     +1 577          MOVX    A, @R0           ; Unroll loop 10
13C0 F0     +1 578          MOVX    @DPTR, A
13C1 A3     +1 579          INC     DPTR
13C2 E2     +1 580          MOVX    A, @R0           ; Unroll loop 11
13C3 F0     +1 581          MOVX    @DPTR, A
13C4 A3     +1 582          INC     DPTR
13C5 E2     +1 583          MOVX    A, @R0           ; Unroll loop 12
13C6 F0     +1 584          MOVX    @DPTR, A
13C7 A3     +1 585          INC     DPTR
13C8 E2     +1 586          MOVX    A, @R0           ; Unroll loop 13
```

```
13C9 F0      +1  587      MOVX   @DPTR, A
13CA A3      +1  588      INC    DPTR
13CB E2      +1  589      MOVX   A, @R0           ; Unroll loop 14
13CC F0      +1  590      MOVX   @DPTR, A
13CD A3      +1  591      INC    DPTR
13CE E2      +1  592      MOVX   A, @R0           ; Unroll loop 15
13CF F0      +1  593      MOVX   @DPTR, A
13D0 A3      +1  594      INC    DPTR
13D1 E2      +1  595      MOVX   A, @R0           ; Unroll loop 16
13D2 F0      +1  596      MOVX   @DPTR, A
13D3 A3      +1  597      INC    DPTR
13D4 DFCE    +1  598      DJNZ   R7, EP2OutLoop
13D6 0555    +1  599      INC    PacketCounter
13D8 78C9    +1  600      MOV    R0, #LOW(EP2OutByteCount)
13DA E2      +1  601      MOVX   A, @R0           ; Get actual Byte Length
13DB F2      +1  602      MOVX   @R0, A           ; Tell SIE we're done with the buffer
13DC B44002  +1  603      CJNE   A, #64, LastOne
13DF 80B1    +1  604      JMP    WaitForPacket    ; Get the next one
13E1         +1  605      LastOne:
13E1 C28C    +1  606      CLR    Timer0_Enable    ; Stop the timer
13E3 F556    +1  607      MOV    LastPacketValidBytes, A
13E5 C2F0    +1  608      CLR    B.0              ; We're done with Task0
13E7 22      +1  609      RET
13E8         +1  610
13E8 908000  +1  611      Task1:                   ; Prepare a BlockIO buffer for PC Host
13EB E4      +1  613      MOV    DPTR, #DataBuffer
13EB E4      +1  613      CLR    A
13EC F58A    +1  614      MOV    Timer0LOW, A     ; Clear Timer 0
13EE F58C    +1  615      MOV    Timer0HIGH, A
13F0 78E3    +1  616      MOV    R0, #LOW(AutoPtrHIGH)
13F2 747E    +1  617      MOV    A, #HIGH(EP2InBuffer)
13F4 F2      +1  618      MOVX   @R0, A
13F5 79B8    +1  619      MOV    R1, #LOW(EP2InStatus)
13F7         +1  620      WaitForBuffer:
13F7 E3      +1  621      MOVX   A, @R1
13F8 20E1FC  +1  622      JB    ACC.1, WaitForBuffer
13FB D28C    +1  623      SETB   Timer0_Enable    ; Time this data xfer operation
13FD 78E4    +1  624      MOV    R0, #LOW(AutoPtrLOW)
13FF 7400    +1  625      MOV    A, #LOW(EP2InBuffer)
1401 F2      +1  626      MOVX   @R0, A
1402 08      +1  627      INC    R0               ; Point to AutoPtr.Data
1403 7F04    +1  628      MOV    R7, #4           ; Assume a full buffer (4*16=64)
1405         +1  629      EP2InLoop:
1405 E0      +1  630      MOVX   A, @DPTR         ; Get data
1406 A3      +1  631      INC    DPTR
1407 F2      +1  632      MOVX   @R0, A           ; Fill buffer. This also bumps ptr
1408 E0      +1  633      MOVX   A, @DPTR         ; Unroll loop 2
1409 A3      +1  634      INC    DPTR
140A F2      +1  635      MOVX   @R0, A
140B E0      +1  636      MOVX   A, @DPTR         ; Unroll loop 3
140C A3      +1  637      INC    DPTR
140D F2      +1  638      MOVX   @R0, A
140E E0      +1  639      MOVX   A, @DPTR         ; Unroll loop 4
140F A3      +1  640      INC    DPTR
1410 F2      +1  641      MOVX   @R0, A
1411 E0      +1  642      MOVX   A, @DPTR         ; Unroll loop 5
1412 A3      +1  643      INC    DPTR
1413 F2      +1  644      MOVX   @R0, A
1414 E0      +1  645      MOVX   A, @DPTR         ; Unroll loop 6
1415 A3      +1  646      INC    DPTR
1416 F2      +1  647      MOVX   @R0, A
1417 E0      +1  648      MOVX   A, @DPTR         ; Unroll loop 7
1418 A3      +1  649      INC    DPTR
1419 F2      +1  650      MOVX   @R0, A
141A E0      +1  651      MOVX   A, @DPTR         ; Unroll loop 8
141B A3      +1  652      INC    DPTR
```

```
141C F2      +1  653      MOVX    @R0, A
141D E0      +1  654      MOVX    A, @DPTR      ; Unroll loop 9
141E A3      +1  655      INC     DPTR
141F F2      +1  656      MOVX    @R0, A
1420 E0      +1  657      MOVX    A, @DPTR      ; Unroll loop 10
1421 A3      +1  658      INC     DPTR
1422 F2      +1  659      MOVX    @R0, A
1423 E0      +1  660      MOVX    A, @DPTR      ; Unroll loop 11
1424 A3      +1  661      INC     DPTR
1425 F2      +1  662      MOVX    @R0, A
1426 E0      +1  663      MOVX    A, @DPTR      ; Unroll loop 12
1427 A3      +1  664      INC     DPTR
1428 F2      +1  665      MOVX    @R0, A
1429 E0      +1  666      MOVX    A, @DPTR      ; Unroll loop 13
142A A3      +1  667      INC     DPTR
142B F2      +1  668      MOVX    @R0, A
142C E0      +1  669      MOVX    A, @DPTR      ; Unroll loop 14
142D A3      +1  670      INC     DPTR
142E F2      +1  671      MOVX    @R0, A
142F E0      +1  672      MOVX    A, @DPTR      ; Unroll loop 15
1430 A3      +1  673      INC     DPTR
1431 F2      +1  674      MOVX    @R0, A
1432 E0      +1  675      MOVX    A, @DPTR      ; Unroll loop 16
1433 A3      +1  676      INC     DPTR
1434 F2      +1  677      MOVX    @R0, A
1435 DFCE     +1  678      DJNZ   R7, EP2InLoop
1437 78B9     +1  679      MOV    R0, #LOW(EP2InByteCount)
1439 D55508   +1  680      DJNZ   PacketCounter, ValidateFullBuffer
143C E556     +1  681      MOV    A, LastPacketValidBytes ; Get actual Byte Length
143E F2      +1  682      MOVX    @R0, A      ; Validate the short buffer
143F C28C     +1  683      CLR    Timer0_Enable ; Stop the timer
1441 C2F1     +1  684      CLR    B.1         ; We're done with Task1
1443 22      +1  685      RET
1444          +1  686      ValidateFullBuffer:
1444 7440     +1  687      MOV    A, #64      ; Default is a full packet
1446 F2      +1  688      MOVX    @R0, A      ; Validate the buffer
1447 80AE     +1  689      JMP    WaitForBuffer ; To prepare more data
              +1  690
1449          +1  691      Wait1second:      ; Give Windows time to unload Driver
1449 7E0A     +1  692      MOV    R6, #10
144B          +1  693      Wait100msec:
144B 7F64     +1  694      MOV    R7, #100
144D          +1  695      Wait1msec:      ; A delay loop
144D 90FB50   +1  696      MOV    DPTR, #-1200 ; Remember that this is DPTR1
1450 A3      +1  697      More:   INC     DPTR ; 3 cycles
1451 E584     +1  698      MOV    A, DPL1    ; + 2
1453 4585     +1  699      ORL   A, DPH1    ; + 2
1455 70F9     +1  700      JNZ   More       ; + 3 = 10 cycles x 1200 = 1msec
1457 DFF4     +1  701      DJNZ  R7, Wait1msec
1459 DEF0     +1  702      DJNZ  R6, Wait100msec
145B 22      +1  703      RET
              +1  704
145C          +1  705      ProcessOutputReport: ; A Report has just been received
              +1  706      ; The report is three bytes long in this example
              +1  707      ; Byte 1 = Command
              +1  708      ;           1 = Read Buttons Value, create Input Report
              +1  709      ;           2 = Update LEDs
              +1  710      ;           Byte 2 = new value
              +1  711      ;           3 = Get ready to receive data
              +1  712      ;           4 = Prepare to send data
              +1  713      ;           Bytes 2 & 3 = data size
145C 907EC0   +1  714      MOV    DPTR, #EP0OutBuffer ; Point to the Report
145F E0      +1  715      MOVX    A, @DPTR    ; Get the Data
1460 A3      +1  716      INC     DPTR        ; Point to data value (if any)
1461 14      +1  717      DEC    A
1462 6023     +1  718      JZ     CreateInputReport
```

```
1464 14      +1 719          DEC      A
1465 7004    +1 720          JNZ     NotUpdateLEDs
1467 E0      +1 721          MOVX   A, @DPTR
1468 F553    +1 722          MOV    LEDValue, A          ; Update the local variable
146A 22      +1 723          RET
146B        +1 724          NotUpdateLEDs:
146B 14      +1 725          DEC      A
146C 7003    +1 726          JNZ     SendCommand
146E        +1 727          ReceiveCommand:
146E D2F0    +1 728          SETB   B.0          ; Setup a task to do
1470 22      +1 729          RET
1471        +1 730          SendCommand:
1471        +1 731          ; Convert BufferSize into PacketCounter and LastPacketValidBytes
1471 E0      +1 732          MOVX   A, @DPTR          ; Low byte of buffer size
1472 543F    +1 733          ANL   A, #00111111b    ; Get Last Packet Bytes
1474 F556    +1 734          MOV    LastPacketValidBytes, A
1476 E0      +1 735          MOVX   A, @DPTR
1477 54C0    +1 736          ANL   A, #11000000b    ; Need top two bits
1479 23      +1 737          RL    A
147A 23      +1 738          RL    A          ; Now in bottom two bits
147B FA      +1 739          MOV    R2, A          ; Save temporary value
147C A3      +1 740          INC    DPTR
147D E0      +1 741          MOVX   A, @DPTR          ; Get high byte of buffer size
147E 23      +1 742          RL    A
147F 23      +1 743          RL    A
1480 4A      +1 744          ORL   A, R2
1481 04      +1 745          INC    A
1482 F555    +1 746          MOV    PacketCounter, A
1484 D2F1    +1 747          SETB   B.1          ; Setup a task to do
1486 22      +1 748          RET
1486        +1 749
1487        +1 750          CreateInputReport:          ; Called from TIMER which detected the need
1487        +1 751          ; The report is only one byte long in this first example
1487        +1 752          ; It contains a new value for the Buttons
1487 907F99  +1 753          MOV    DPTR, #PortA_Pins
148A E0      +1 754          MOVX   A, @DPTR
148B 907E80  +1 755          MOV    DPTR, #EP1InBuffer    ; Point to the buffer
148E F0      +1 756          MOVX   @DPTR, A          ; Update the Report
148F 907FB7  +1 757          MOV    DPTR, #EP1InByteCount
1492 7401    +1 758          MOV    A, #1
1494 F0      +1 759          MOVX   @DPTR, A          ; Endpoint 1 now 'armed', next IN will get data
1495 22      +1 760          RET
1495        +1 761
1495        +1 762
1495        +1 763          ;$include (Decode.A51)
1495        +1 764          ; This module is common to all of the examples.
1495        +1 765          ; It decodes the USB Setup Packets and generates appropriate responses.
1495        +1 766          ; Interpretation of Reports is handled by MAIN
1495        +1 767          ;
----      +1 768          CSEG
1496        +1 769          ServiceSetupPacket:
1496 E549    +1 770          MOV    A, RequestType
1498 A2E7    +1 771          MOV    C, ACC.7          ; Bit 7 = 1 means IO device needs to send data
to P
149A 9202    +1 772          C Host
149C 545C    +1 773          MOV    SendData, C
149E 7035    +1 774          ANL   A, #01011100b    ; IF RequestType[6.4.3.2] = 1 THEN goto
BadRequest
14A0 E549    +1 775          JNZ     BadRequest
14A2 A2E0    +1 776          MOV    A, RequestType    ; IF RequestType[1&0] = 1 THEN goto BadRequest
14A4 82E1    +1 777          MOV    C, ACC.0
14A6 402D    +1 778          ANL   C, ACC.1
14A8 30E502 +1 779          JC     BadRequest
[1,
1]
14AB 7403    +1 780          JNB   ACC.5, NotB5      ; IF RequestType[5] = 1 THEN RequestType[1,0] =
14AD 5403    +1 781          NotB5: ANL   A, #00000011b    ; Set CommandIndex[5,4] = RequestType[1,0]
14AF C4      +1 782          SWAP  A
```

```
14B0 FF      +1 783      MOV     R7, A           ; Save HI nibble of CommandIndex
              +1 784                      ; Set CommandIndex[3,0] = Request[3,0]
14B1 E54A    +1 785      MOV     A, Request
14B3 54F0    +1 786      ANL    A, #11110000b   ; Check if Request > 15
14B5 701E    +1 787      JNZ    BadRequest
14B7 E54A    +1 788      MOV     A, Request
14B9 540F    +1 789      ANL    A, #00001111b   ; Only 13 are defined today, handle in table
14BB 4F      +1 790      ORL    A, R7
              +1 791      ; CALL   CorrectSubroutine   ; goto CommandTable(CommandIndex)
14BC         +1 792      CorrectSubroutine:   ; Jump to the subroutine that DPTR is pointing
to
14BC 754001  +1 793      MOV     ReplyCount, #1   ; Set up a default reply
14BF 754100  +1 794      MOV     ReplyBuffer, #0
14C2 754200  +1 795      MOV     ReplyBuffer+1, #0
14C5 C204    +1 796      CLR    SetAddress       ; Clear all flags
14C7 C201    +1 797      CLR    STALL
14C9 C203    +1 798      CLR    IsDescriptor
14CB 9014E2  +1 799      MOV     DPTR, #CommandTable
14CE 91D9    +1 800      CALL   BumpDPTR        ; Point to entry
14D0 E0      +1 801      MOVX   A, @DPTR        ; Get the offset
14D1 901522  +1 802      MOV     DPTR, #Subroutines
14D4 73      +1 803      JMP    @A+DPTR         ; Go to the correct Subroutine
              +1 804
14D5         +1 805      BadRequest:          ; Decoded a Bad Request, STALL the Endpoint
14D5 D201    +1 806      SETB   STALL
14D7 22      +1 807      RET
              +1 808
14D8         +1 809      NextDPTR:           ; Returns (DPTR + byte DPTR is pointing to)
14D8 E0      +1 810      MOVX   A, @DPTR
14D9         +1 811      BumpDPTR:          ; Returns (DPTR + ACC)
14D9 2582    +1 812      ADD    A, DPL
14DB F582    +1 813      MOV    DPL, A
14DD 5002    +1 814      JNC    Skip
14DF 0583    +1 815      INC    DPH           ; Need 16 bit arithmetic here
14E1 22      +1 816      Skip:   RET
              +1 817
              +1 818      ; Since the table only contains byte offsets, it is important that all these routines
are
              +1 819      ; within one page (100H) of Subroutines
              +1 820      ; V3.0 - CommandTable moved outside of this one page limited space
14E2         +1 821      CommandTable:
              +1 822      ; First 16 commands are for the Device
14E2 1B      +1 823      DB LOW(Device_Get_Status - Subroutines)
14E3 00      +1 824      DB LOW(Device_Clear_Feature - Subroutines)
14E4 00      +1 825      DB LOW(Invalid - Subroutines)
14E5 00      +1 826      DB LOW(Device_Set_Feature - Subroutines)
14E6 00      +1 827      DB LOW(Invalid - Subroutines)
14E7 03      +1 828      DB LOW(Set_Address - Subroutines)
14E8 37      +1 829      DB LOW(Get_Descriptor - Subroutines)
14E9 00      +1 830      DB LOW(Set_Descriptor - Subroutines)
14EA 17      +1 831      DB LOW(Get_Configuration - Subroutines)
14EB 23      +1 832      DB LOW(Set_Configuration - Subroutines)
14EC 00      +1 833      DB LOW(Invalid - Subroutines)
14ED 00      +1 834      DB LOW(Invalid - Subroutines)
14EE 00      +1 835      DB LOW(Invalid - Subroutines)
14EF 00      +1 836      DB LOW(Invalid - Subroutines)
14F0 00      +1 837      DB LOW(Invalid - Subroutines)
14F1 00      +1 838      DB LOW(Invalid - Subroutines)
              +1 839      ; Next 16 commands are for the Interface
14F2 1F      +1 840      DB LOW(Interface_Get_Status - Subroutines)
14F3 00      +1 841      DB LOW(Interface_Clear_Feature - Subroutines)
14F4 00      +1 842      DB LOW(Invalid - Subroutines)
14F5 00      +1 843      DB LOW(Interface_Set_Feature - Subroutines)
14F6 00      +1 844      DB LOW(Invalid - Subroutines)
14F7 00      +1 845      DB LOW(Invalid - Subroutines)
14F8 5E      +1 846      DB LOW(Get_Class_Descriptor - Subroutines)
14F9 00      +1 847      DB LOW(Set_Class_Descriptor - Subroutines)
14FA 00      +1 848      DB LOW(Invalid - Subroutines)
```

```

14FB 00      +1  849          DB LOW(Invalid - Subroutines)
14FC 00      +1  850          DB LOW(Get_Interface - Subroutines)
14FD 00      +1  851          DB LOW(Set_Interface - Subroutines)
14FE 00      +1  852          DB LOW(Invalid - Subroutines)
14FF 00      +1  853          DB LOW(Invalid - Subroutines)
1500 00      +1  854          DB LOW(Invalid - Subroutines)
1501 00      +1  855          DB LOW(Invalid - Subroutines)
1502 1F      +1  856          ; Next 16 commands are for the Endpoint
1503 00      +1  857          DB LOW(Endpoint_Get_Status - Subroutines)
1504 00      +1  858          DB LOW(Endpoint_Clear_Feature - Subroutines)
1505 00      +1  859          DB LOW(Invalid - Subroutines)
1506 00      +1  860          DB LOW(Endpoint_Set_Feature - Subroutines)
1507 00      +1  861          DB LOW(Invalid - Subroutines)
1508 00      +1  862          DB LOW(Invalid - Subroutines)
1509 00      +1  863          DB LOW(Invalid - Subroutines)
150A 00      +1  864          DB LOW(Invalid - Subroutines)
150B 00      +1  865          DB LOW(Invalid - Subroutines)
150C 00      +1  866          DB LOW(Invalid - Subroutines)
150D 00      +1  867          DB LOW(Invalid - Subroutines)
150E 00      +1  868          DB LOW(Invalid - Subroutines)
150F 00      +1  869          DB LOW(Endpoint_Sync_Frame - Subroutines)
1510 00      +1  870          DB LOW(Invalid - Subroutines)
1511 00      +1  871          DB LOW(Invalid - Subroutines)
1512 00      +1  872          DB LOW(Invalid - Subroutines)
1513 0D      +1  873          ; Next 16 commands are Class Requests
1514 00      +1  874          DB LOW(Invalid - Subroutines)
1515 00      +1  875          DB LOW(Get_Report - Subroutines)
1516 00      +1  876          DB LOW(Get_Idle - Subroutines)
1517 00      +1  877          DB LOW(Get_Protocol - Subroutines)
1518 00      +1  878          DB LOW(Invalid - Subroutines)
1519 00      +1  879          DB LOW(Invalid - Subroutines)
151A 00      +1  880          DB LOW(Invalid - Subroutines)
151B 06      +1  881          DB LOW(Invalid - Subroutines)
151C 00      +1  882          DB LOW(Invalid - Subroutines)
151D 00      +1  883          DB LOW(Set_Report - Subroutines)
151E 00      +1  884          DB LOW(Set_Idle - Subroutines)
151F 00      +1  885          DB LOW(Set_Protocol - Subroutines)
1520 00      +1  886          DB LOW(Invalid - Subroutines)
1521 00      +1  887          DB LOW(Invalid - Subroutines)
1522         +1  888          DB LOW(Invalid - Subroutines)
1522         +1  889          DB LOW(Invalid - Subroutines)
1522         +1  890          DB LOW(Invalid - Subroutines)
1522         +1  891          Subroutines:
1522         +1  892          ;
1522         +1  893          ; Many requests are INVALID for this example
1522         +1  894          Get_Protocol:                ; We are not a Boot device
1522         +1  895          Set_Protocol:                ; We are not a Boot device
1522         +1  896          Set_Descriptor:              ; Our Descriptors are static
1522         +1  897          Set_Class_Descriptor:        ; Our Descriptors are static
1522         +1  898          Set_Interface:              ; We only have one Interface
1522         +1  899          Get_Interface:              ; We do not have an Alternate setting
1522         +1  900          Set_Idle:                    ; V3.0 Optional command, not supported
1522         +1  901          Get_Idle:                    ; V3.0 Optional command, not supported
1522         +1  902          Device_Set_Feature:          ; We have no features that can be set or cleared
1522         +1  903          Interface_Set_Feature:       ; We have no features that can be set or cleared
1522         +1  904          Endpoint_Set_Feature:        ; We have no features that can be set or cleared
1522         +1  905          Endpoint_Clear_Feature:     ; V3.0 We have no features that can be set or cleared
1522         +1  906          Device_Clear_Feature:       ; We have no features that can be set or cleared
1522         +1  907          Interface_Clear_Feature:    ; We have no features that can be set or cleared
1522         +1  908          Endpoint_Sync_Frame:        ; We are not an Isonchronous device
1522         +1  909          ;
1522         +1  910          Invalid:                      ; Invalid Request made, STALL the Endpoint
1522 D201     +1  911          SETB      STALL
1524 22      +1  912          Reply:    RET
1525         +1  913          ;
1525         +1  914          Set_Address:                ; Set the address that the SIE will respond to

```

```
1525 D204      +1  915          SETB   SetAddress
1527 22        +1  916          RET
                +1  917
1528          +1  918      Set_Report:          ; Host wants to sent us a Report.
                +1  919      ; The ONLY case in this example where host sends data to us
1528 3000F7    +1  920          JNB     Configured, Invalid ; Need to be Configured to do this command
152B 7102     +1  921          CALL   GetOutputReport ; Handled in EZUSB.A51
152D 815C     +1  922          JMP     ProcessOutputReport ; RETurn via this subroutine
152F          +1  923      Get_Report:          ; Host wants a Report
152F 3000F0    +1  924          JNB     Configured, Invalid ; Need to be Configured to do this command
1532 907F99   +1  925          MOV     DPTR, #PortA_Pins ; Get Buttons value
1535 E0       +1  926          MOVX   A, @DPTR
1536 F541     +1  927          MOV     ReplyBuffer, A
1538 22       +1  928          RET
1539          +1  929      Get_Configuration: ; Respond with CurrentConfiguration
1539 854341    +1  930          MOV     ReplyBuffer, CurrentConfiguration
153C 22       +1  931          RET
153D          +1  932      Device_Get_Status: ; Only two bits of Device Status are defined
153D 754101   +1  933          MOV     ReplyBuffer, #1 ; Bit 1=Remote Wakeup(=0), Bit 0=Self
Powered(=1)
1540 22       +1  934          RET
1541          +1  935      Interface_Get_Status: ; Interface Status is currently defined as 0
1541          +1  936      Endpoint_Get_Status:
1541 754002    +1  937          MOV     ReplyCount, #2 ; Need a two byte 0 response
1544 22       +1  938          RET
1545          +1  939      Set_Configuration: ; Valid values are 0 and 1
1545 E54B     +1  940          MOV     A, wValueLow
1547 600A     +1  941          JZ     Deconfigured
1549 14       +1  942          DEC     A
154A 70D6     +1  943          JNZ   Invalid
154C D200     +1  944          SETB   Configured
154E 754301   +1  945          MOV     CurrentConfiguration, #1
                +1  946      ; Complete any other application specific initialization
1551 6178     +1  947          JMP     EnableDevice
1553          +1  948      Deconfigured:
1553 C200     +1  949          CLR     Configured
1555 F543     +1  950          MOV     CurrentConfiguration, A
                +1  951      ; Complete any other application specific disabling
1557 6179     +1  952          JMP     DisableDevice
1559          +1  953      Get_Descriptor: ; Host wants to know who/what we are
1559 D203     +1  954          SETB   IsDescriptor
155B E54C     +1  955          MOV     A, wValueHigh
155D 14       +1  956          DEC     A ; Valid Values are 1, 2 and 3
155E 901599   +1  957          MOV     DPTR, #DeviceDescriptor
1561 6031     +1  958          JZ     ReturnLength
1563 14       +1  959          DEC     A
1564 9015AB   +1  960          MOV     DPTR, #ConfigurationDescriptor
1567 7003     +1  961          JNZ   TryString
1569 7439     +1  962          MOV     A, #ConfigLength
156B 22       +1  963          RET
156C          +1  964      TryString:
156C 14       +1  965          DEC     A
156D 70B3     +1  966          JNZ   Invalid
                +1  967      ; Request is for a String Descriptor
156F 901602   +1  968          MOV     DPTR, #String0 ; Point to String 0
1572 E54B     +1  969          MOV     A, wValueLow ; Get String Index
1574          +1  970      NextString:
1574 601E     +1  971          JZ     ReturnLength
1576 FF       +1  972          MOV     R7, A ; Save String Index
1577 91D8     +1  973          CALL   NextDPTR
1579 E0       +1  974          MOVX   A, @DPTR ; Get the String Length (= 0 means we're at
Backsto
                +1  975          p)
157A 60A6     +1  975          JZ     Invalid ; Asked for a string I don't have
157C EF       +1  976          MOV     A, R7
157D 14       +1  977          DEC     A
157E 80F4     +1  978          JMP     NextString ; Check if we are there yet
1580          +1  979      Get_Class_Descriptor: ; Valid values are 21H, 22H, 23H for Class
Request
```

```
1580 D203      +1  980          SETB   IsDescriptor
1582 E54C      +1  981          MOV    A, wValueHigh
1584 C3        +1  982          CLR    C
1585 9421      +1  983          SUBB   A, #21H
1587 9015BD    +1  984          MOV    DPTR, #HIDDescriptor
158A 6008      +1  985          JZ     ReturnLength
158C 14        +1  986          DEC    A
158D 9015E4    +1  987          MOV    DPTR, #ReportDescriptor
1590 6004      +1  988          JZ     ReturnRDlength
              +1  989          ;     DEC    A                      ; This example does not use Physical
Descriptors
              +1  990          ;     JZ     Send_Physical_Descriptor
1592 808E      +1  991          JMP    Invalid
              +1  992          ;
1594           +1  993          ReturnLength:
1594 E0        +1  994          MOVX   A, @DPTR                      ; Get Descriptor Length (first byte)
1595 22        +1  995          RET
1596           +1  996          ReturnRDlength:                ; Report Descriptor is different format
1596 741E      +1  997          MOV    A, #ReportLength
1598 22        +1  998          RET
              +1  999          ; Error check: this MUST be on within a page of Subroutines
0077          +1 1000          WithinSamePage EQU $ - Subroutines
              +1 1001          ;
              +1 1002          ;
              +1 1003          ;
              +1 1004          ;$include (DTables.A51)
              +1 1005          ; This module declares the descriptors
              +1 1006          ;
              +1 1007          ; This example has one Device Descriptor with:
              +1 1008          ;   One Configuration
              +1 1009          ;   Two Interfaces:
              +1 1010          ;       HID:   One HID Descriptor - to make PC host software simpler
              +1 1011          ;           One Endpoint Descriptor - for HID Input Reports
              +1 1012          ;           One Report Descriptor - one byte IN and one byte OUT reports
              +1 1013          ;       BlockIO: Two Endpoint Descriptors
              +1 1014          ;       Multiple Sting Descriptors - to aid the user
              +1 1015          ;
              +1 1016          CSEG
1599          +1 1017          DeviceDescriptor:
1599 1201      +1 1018          DB    18, 1                      ; Length, Type
159B 1001      +1 1019          DB    10H, 1                      ; USB Rev 1.1 (=0110H, low=10H, High=01H)
159D 000000    +1 1020          DB    0, 0, 0                      ; Class, Subclass and Protocol
15A0 40        +1 1021          DB    EPOSize
15A1 42422042 +1 1022          DB    42H, 42H, 20H, 42H, 0, 1; Vendor ID, Product ID and Version
15A5 0001
15A7 010200    +1 1023          DB    1, 2, 0                      ; Manufacturer, Product & Serial# Names
15AA 01        +1 1024          DB    1                      ; #Configs
15AB          +1 1025          ConfigurationDescriptor:
15AB 0902      +1 1026          DB    9, 2                      ; Length, Type
15AD 3900      +1 1027          DB    LOW(ConfigLength), HIGH(ConfigLength)
15AF 020103    +1 1028          DB    2, 1, 3                      ; #Interfaces, Configuration#, Config. Name
15B2 80        +1 1029          DB    10000000b                ; Attributes = Bus Powered
15B3 FA        +1 1030          DB    250                      ; Max. Power is 250x2 = 500mA
15B4          +1 1031          HIDInterfaceDescriptor:
15B4 0904      +1 1032          DB    9, 4                      ; Length, Type
15B6 000001    +1 1033          DB    0, 0, 1                  ; ID, No alternate setting, HID uses EP1 In
15B9 03        +1 1034          DB    3                      ; Class = Human Interface Device
15BA 0000      +1 1035          DB    0, 0                      ; Subclass and Protocol
15BC 04        +1 1036          DB    4                      ; Interface Name
15BD          +1 1037          HIDDescriptor:
15BD 0921      +1 1038          DB    9, 21H                  ; Length, Type
15BF 0001      +1 1039          DB    0, 1                      ; HID Class Specification compliance
15C1 00        +1 1040          DB    0                      ; Country localization (=none)
15C2 01        +1 1041          DB    1                      ; Number of descriptors to follow
15C3 22        +1 1042          DB    22H                      ; And it's a Report descriptor
15C4 1E00      +1 1043          DB    LOW(ReportLength), HIGH(ReportLength)
15C6          +1 1044          HIDEndpointDescriptor:
```

```

15C6 0705      +1 1045      DB      7, 5          ; Length, Type
15C8 81        +1 1046      DB      10000001b    ; Address = IN 1
15C9 03        +1 1047      DB      00000011b    ; Interrupt
15CA 4000      +1 1048      DB      EPOSize, 0   ; Maximum packet size (this example only uses 1)
15CC 0A        +1 1049      DB      10           ; Poll every 10msec seconds
15CD          +1 1050      BlockIOInterfaceDescriptor:
15CD 0904      +1 1051      DB      9, 4          ; Length, Type
15CF 010002    +1 1052      DB      1, 0, 2      ; ID, No alternate setting, BlockIO uses EP2 In/Out
15D2 FF        +1 1053      DB      0FFH         ; Class = Vendor Defined (needs BlockIO.sys)
15D3 0000      +1 1054      DB      0, 0         ; Subclass and Protocol
15D5 05        +1 1055      DB      5            ; Interface Name
15D6          +1 1056      BlockIOInEndpointDescriptor:
15D6 0705      +1 1057      DB      7, 5          ; Length, Type
15D8 82        +1 1058      DB      10000010b    ; Address = IN 2
15D9 02        +1 1059      DB      00000010b    ; Bulk
15DA 4000      +1 1060      DB      EPOSize, 0   ; Maximum packet size
15DC 00        +1 1061      DB      0            ; Ignorred
15DD          +1 1062      BlockIOOutEndpointDescriptor:
15DD 0705      +1 1063      DB      7, 5          ; Length, Type
15DF 02        +1 1064      DB      00000010b    ; Address = OUT 2
15E0 02        +1 1065      DB      00000010b    ; Bulk
15E1 4000      +1 1066      DB      EPOSize, 0   ; Maximum packet size
15E3 00        +1 1067      DB      0            ; Ignorred
   0039        +1 1068      ConfigLength      EQU $ - ConfigurationDescriptor
            +1 1069
15E4          +1 1070      ReportDescriptor:    ; Generated with HID Tool, copied to here
15E4 0600FF    +1 1071      DB      6, 0, 0FFH   ; Usage_Page (Vendor Defined)
15E7 0901      +1 1072      DB      9, 1         ; Usage (I/O Device)
15E9 A101      +1 1073      DB      0A1H, 1      ; Collection (Application)
15EB 1901      +1 1074      DB      19H, 1       ; Usage_Minimum (User Defined)
15ED 2908      +1 1075      DB      29H, 8       ; Usage_Maximum (User Defined)
15EF 1580      +1 1076      DB      15H, 80H     ; Logical_Minimum (-128)
15F1 257F      +1 1077      DB      25H, 7FH     ; Logical_Maximum (127)
15F3 7508      +1 1078      DB      75H, 8       ; Report_Size (8)
15F5 9501      +1 1079      DB      95H, 1       ; Report_Count (1)
15F7 8102      +1 1080      DB      81H, 2       ; Input (Data,Var,Abs)
15F9 1901      +1 1081      DB      19H, 1       ; Usage_Minimum (User Defined)
15FB 2908      +1 1082      DB      29H, 8       ; Usage_Maximum (User Defined)
15FD 9503      +1 1083      DB      95H, 3       ; Report_Count (3)
15FF 9102      +1 1084      DB      91H, 2       ; Output (Data,Var,Abs)
1601 C0        +1 1085      DB      0C0H         ; End_Collection
   001E        +1 1086      ReportLength      EQU $-ReportDescriptor
            +1 1087
1602          +1 1088      String0:           ; Declare the UNICODE strings
1602 04030904  +1 1089      DB      4, 3, 9, 4   ; Only English language strings supported
1606          +1 1090      String1:           ; Manufacturer
1606 2C03      +1 1091      DB      (String2-String1),3 ; Length, Type
1608 55005300  +1 1092      DB      "U",0,"S",0,"B",0," ",0,"D",0,"e",0,"s",0,"i",0,"g",0,"n",0," ",0
160C 42002000
1610 44006500
1614 73006900
1618 67006E00
161C 2000
161E 42007900  +1 1093      DB      "B",0,"y",0," ",0,"E",0,"x",0,"a",0,"m",0,"p",0,"l",0,"e",0
1622 20004500
1626 78006100
162A 6D007000
162E 6C006500
1632          +1 1094      String2:           ; Product Name
1632 0C03      +1 1095      DB      (String3-String2),3
1634 45004300  +1 1096      DB      "E",0,"C",0,"H",0,"O",0,"l",0
1638 48004F00
163C 3100
163E          +1 1097      String3:           ; Configuration Name
163E 1E03      +1 1098      DB      (String4-String3),3
1640 48004900  +1 1099      DB      "H",0,"I",0,"D",0,"w",0,"i",0,"t",0,"h",0

```

```
1644 44007700
1648 69007400
164C 6800
164E 42006C00 +1 1100          DB      "B",0,"l",0,"o",0,"c",0,"k",0,"I",0,"O",0
1652 6F006300
1656 6B004900
165A 4F00
165C          +1 1101      String4:          ; Interface 1 Name
165C 0803      +1 1102          DB      (String5-String4),3
165E 48004900 +1 1103          DB      "H",0,"I",0,"D",0
1662 4400
1664          +1 1104      String5:          ; Interface 2 Name
1664 1003      +1 1105          DB      (EndOfDescriptors-String5),3
1666 42006C00 +1 1106          DB      "B",0,"l",0,"o",0,"c",0,"k",0,"I",0,"O",0
166A 6F006300
166E 6B004900
1672 4F00
1674          +1 1107      EndOfDescriptors:
1674 00        +1 1108          DB      0          ; Backstop for String Descriptors
          +1 1109
          +1 1110
          +1 1111
          1112
          1113
          1114
          1115      END
```