

MACRO ASSEMBLER A51 V6.10

OBJECT MODULE PLACED IN .\BAL.OBJ

ASSEMBLER INVOKED BY: C:\Keil\C51\BIN\A51.EXE .\BAL.A51 REGISTERBANK(0) SET(SMALL) DEBUG EP

```

LOC OBJ          LINE    SOURCE
                1      NAME    ButtonsAndLights
                2      ; Version 3.5
                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      ; This version works with dScope monSIO0.hex (uses Serial Port 0, loads at 1200H)
               14      ;
               15      ; Changes from Version 3.0
               16      ;     a) USBReset interrupt was not enabled! It now is (Thank you Greg Porter)
               17      ;     b) No reports are generated unless the device is Configured
               18      ;     c) Retrieve LENGTH missing near LoadEP0 (Thank you Greg Porter)
               19      ;     d) Windows given 200msec to unload driver during renumerate
               20      ;
               21      ; Changes from Version 2.0
               22      ;     a) Two misplaced labels corrected
               23      ;     b) CommandTable moved outside of the constrained page
               24      ;     c) SETUPDAT buffer copied to direct access memory, simplified coding
               25      ;     d) Optional Set_Idle now not supported (returns STALL, not ignorred)
               26      ;     e) R7 used in place of Temp, saved code space
               27      ;     f) Code reorganized to separate hardware dependant sections
               28      ;         Old          New
               29      ;         USBINT.A51    Decode.A51
               30      ;         Vectors.A51    EZInt.A51
               31      ;         Timer.A51      EZInt.A51
               32      ;         Main.A51      EZMain.A51
               33      ;     g) EP0Size made an equate to ease coding of other components
0040          34      EP0Size EQU    64      ; For EZ-USB
               35      ;     h) Code added for descriptors > EP0Size
               36      ;
               37      ; Changes from Version 1.0
               38      ;     a) Register saving removed from Vectors.A51
               39      ;         Main has no context which needs to be saved
               40      ;     b) There was a race condition in USB_INT::Set_Report: which
               41      ;         could cause OLD data to be read. Busy algorithm changed
               42      ;     c) The USB Version# was incorrectly declared in the Device Descriptor
               43      ;         It was 0101H and is now 0110H (data from USB IF)
               44      ;
               45      ;$include (Declare.A51)
               +1      46      ; This module declares the variables and constants used in the examples
               +1      47      ; It is common to all of the examples
               +1      48      ;
               +1      49      ; Declare Special Function Registers used
0088          +1      50      TimerControl    DATA    088H
0089          +1      51      TimerMode      DATA    089H
008C          +1      52      Timer0High     DATA    08CH
00A8          +1      53      EI              DATA    0A8H
00E8          +1      54      EIE           DATA    0E8H      ; EZ-USB specific
0091          +1      55      EXIF         DATA    091H      ; EZ-USB specific
00D8          +1      56      EICON        DATA    0D8H      ; EZ-USB specific
0092          +1      57      PageReg       DATA    092H      ; EZ-USB specific, used with MOVX @Ri
0086          +1      58      DPS           DATA    086H      ; EZ-USB specific, used with dual data pointers

```

```

+1 59 ;
+1 60 ; "External" memory locations used, EZ-USB specific
+1 61 ; Note that most of these variables are in Page 7FH
7FE8 +1 62 SETUPDAT EQU 07FE8H
7FD4 +1 63 SUDPTR EQU 07FD4H
7FB4 +1 64 EP0Control EQU 07FB4H
7F00 +1 65 EP0InBuffer EQU 07F00H
7EC0 +1 66 EP0OutBuffer EQU 07EC0H ; Not in Page 7FH
7E80 +1 67 EP1InBuffer EQU 07E80H ; Not in Page 7FH
7FB5 +1 68 IN0ByteCount EQU 07FB5H
7FC5 +1 69 Out0ByteCount EQU 07FC5H
7FB7 +1 70 IN1ByteCount EQU 07FB7H
7FAC +1 71 IN07IEN EQU 07FACH
7FA9 +1 72 IN07IRQ EQU 07FA9H
7FAD +1 73 OUT07IEN EQU 07FADH
7FAA +1 74 OUT07IRQ EQU 07FAAH
7FAE +1 75 USBIEN EQU 07FAEH
7FAB +1 76 USBIRQ EQU 07FABH
7FD6 +1 77 USBControl EQU 07FD6H
7FA6 +1 78 I2CData EQU 07FA6H
7FA5 +1 79 I2CControl EQU 07FA5H
7F93 +1 80 PortA_Config EQU 07F93H
7F94 +1 81 PortB_Config EQU 07F94H
7F95 +1 82 PortC_Config EQU 07F95H
7F96 +1 83 PortA_OUT EQU 07F96H
7F97 +1 84 PortB_OUT EQU 07F97H
7F98 +1 85 PortC_OUT EQU 07F98H
7F99 +1 86 PortA_PINS EQU 07F99H
7F9A +1 87 PortB_PINS EQU 07F9AH
7F9B +1 88 PortC_PINS EQU 07F9BH
7F9C +1 89 PortA_OE EQU 07F9CH
7F9D +1 90 PortB_OE EQU 07F9DH
7F9E +1 91 PortC_OE EQU 07F9EH
+1 92 ;
+1 93 ; Byte Variables
+1 94
---- +1 95 DSEG AT 20H
0020 +1 96 FLAGS: DS 1 ; This register is bit-addressable
+1 97 ; Bit Variables
0000 +1 98 Configured EQU FLAGS.0 ; Is this device configured
0001 +1 99 STALL EQU FLAGS.1 ; Need to STALL endpoint 0
0002 +1 100 SendData EQU FLAGS.2 ; Need to send data to PC Host
0003 +1 101 IsDescriptor EQU FLAGS.3 ; Enable a shortcut reply
0004 +1 102 SetAddress EQU FLAGS.4 ; Set the SIE address
+1 103 ;
0021 +1 104 MonitorSpace: DS 1FH ; Used by Dscope
0040 +1 105 DataSpace:
0040 +1 106 ReplyCount: DS 1 ; Byte count for following buffer
0041 +1 107 ReplyBuffer: DS 2 ; Buffer for immediate reply
0043 +1 108 CurrentConfiguration:
0043 +1 109 DS 1 ; Some examples support > 1 configurations
0044 +1 110 SaveDPH: DS 1 ; Needed to save Descriptor Pointer ..
0045 +1 111 SaveDPL: DS 1 ; .. for descriptors > EP0Size
0046 +1 112 SaveLength: DS 1 ; Number of bytes still to send
0047 +1 113 SetupData: ; Buffer in direct access memory
0047 +1 114 RequestType: DS 1
0048 +1 115 Request: DS 1
0049 +1 116 wValueLow: DS 1
004A +1 117 wValueHigh: DS 1
004B +1 118 wIndexLow: DS 1
004C +1 119 wIndexHigh: DS 1
004D +1 120 wLengthLow: DS 1
004E +1 121 wLengthHigh: DS 1
+1 122 ;
004F +1 123 Old_Buttons: DS 1 ; Used by BAL: stores current button position
0050 +1 124 LEDstrobe: DS 1 ; Used by BAL: strobe one LED on at a time

```

```
0051      +1 125      LEDvalue:      DS      1      ; Used by BAL: stores current LED value
0052      +1 126      Msec_Counter:  DS      1      ; Used by BAL: counts up to 4 msec
          +1 127      ;
          128
          129      ;$include (EZInt.A51)
          +1 130      ; This module contains all the EZUSB-specific hardware code
          +1 131      ; This module also contains all of the interrupt vector declarations and
          +1 132      ; the first level interrupt servicing (register save, call subroutine,
          +1 133      ; clear interrupt source, restore registers, return)
          +1 134      ; Suspend and Resume are handled totally in this module
          +1 135      ;
          +1 136      ; A Reset sends us to Program space location 0
-----  +1 137      CSEG AT 0      ; Code space
          +1 138      USING 0      ; Reset forces Register Bank 0
0000 021308 +1 139      LJMP      Reset
          +1 140      ;
          +1 141      ; The interrupt vector table is also located here
          +1 142      ; EZ-USB has two levels of USB interrupts:
          +1 143      ; 1-the main level is described in this table (at ORG 43H)
          +1 144      ; 2-there are 21 sources of USB interrupts and these are described in USB_ISR
          +1 145      ; This means that two levels of acknowledgement and clearing will be required
          +1 146      ;      LJMP      INT0_ISR      ; Features not used are commented out
          +1 147      ;      ORG      0BH
          +1 148      ;      LJMP      Timer0_ISR
          +1 149      ;      ORG      13H
          +1 150      ;      LJMP      INT1_ISR
          +1 151      ;      ORG      1BH
          +1 152      ;      LJMP      Timer1_ISR
          +1 153      ;      ORG      23H
          +1 154      ;      LJMP      UART0_ISR
          +1 155      ;      ORG      2BH
          +1 156      ;      LJMP      Timer2_ISR
0033      +1 157      ORG      33H
0033 021245 +1 158      LJMP      WakeUp_ISR
          +1 159      ;      ORG      3BH
          +1 160      ;      LJMP      UART1_ISR
0043      +1 161      ORG      43H
0043 021200 +1 162      LJMP      USB_ISR      ; Auto Vector will replace byte 45H
          +1 163      ;      ORG      4BH
          +1 164      ;      LJMP      I2C_ISR
          +1 165      ;      ORG      53H
          +1 166      ;      LJMP      INT4_ISR
          +1 167      ;      ORG      5BH
          +1 168      ;      LJMP      INT5_ISR
          +1 169      ;      ORG      63H
          +1 170      ;      LJMP      INT6_ISR
          +1 171      ;
1200      +1 172      ORG      1200H      ; Load above monSIO0.hex
1200 021283 +1 173      USB_ISR:LJMP  SUDAV_ISR
1203 00      +1 174      DB      0      ; Pad entries to 4 bytes
1204 02126C +1 175      LJMP      SOF_ISR
1207 00      +1 176      DB      0
1208 02121B +1 177      LJMP      SUTOK_ISR
120B 00      +1 178      DB      0
120C 021230 +1 179      LJMP      Suspend_ISR
120F 00      +1 180      DB      0
1210 021223 +1 181      LJMP      USBReset_ISR
1213 00      +1 182      DB      0
1214 02121B +1 183      LJMP      Reserved
1217 00      +1 184      DB      0
1218 02125B +1 185      LJMP      EP0In_ISR
          +1 186      ;      DB      0      ; Comment out features not used
          +1 187      ;      LJMP      EP0Out_ISR
          +1 188      ;      DB      0
          +1 189      ;      LJMP      EP1In_ISR
          +1 190      ;      DB      0
```

```
+1 191 ; LJMP EP1Out_ISR
+1 192 ; DB 0
+1 193 ; LJMP EP2In_ISR
+1 194 ; DB 0
+1 195 ; LJMP EP2Out_ISR
+1 196 ; DB 0
+1 197 ; LJMP EP3In_ISR
+1 198 ; DB 0
+1 199 ; LJMP EP3Out_ISR
+1 200 ; DB 0
+1 201 ; LJMP EP4In_ISR
+1 202 ; DB 0
+1 203 ; LJMP EP4Out_ISR
+1 204 ; DB 0
+1 205 ; LJMP EP5In_ISR
+1 206 ; DB 0
+1 207 ; LJMP EP5Out_ISR
+1 208 ; DB 0
+1 209 ; LJMP EP6In_ISR
+1 210 ; DB 0
+1 211 ; LJMP EP6Out_ISR
+1 212 ; DB 0
+1 213 ; LJMP EP7In_ISR
+1 214 ; DB 0
+1 215 ; LJMP EP7Out_ISR
+1 216 ; End of Interrupt Vector tables
+1 217
+1 218 ; When a feature is used insert the required interrupt processing here
+1 219 ; The example use only used Endpoints 0 and 1 and also SOF for timing
121B +1 220 Reserved:
121B +1 221 INT0_ISR:
121B +1 222 Timer0_ISR:
121B +1 223 INT1_ISR:
121B +1 224 Timer1_ISR:
121B +1 225 UART0_ISR:
121B +1 226 Timer2_ISR:
121B +1 227 UART1_ISR:
121B +1 228 I2C_ISR:
121B +1 229 INT4_ISR:
121B +1 230 INT5_ISR:
121B +1 231 INT6_ISR:
121B +1 232 SUTOK_ISR:
121B +1 233 EP0Out_ISR:
121B +1 234 EP1In_ISR:
121B +1 235 EP1Out_ISR:
121B +1 236 EP2In_ISR:
121B +1 237 EP2Out_ISR:
121B +1 238 EP3In_ISR:
121B +1 239 EP3Out_ISR:
121B +1 240 EP4In_ISR:
121B +1 241 EP4Out_ISR:
121B +1 242 EP5In_ISR:
121B +1 243 EP5Out_ISR:
121B +1 244 EP6In_ISR:
121B +1 245 EP6Out_ISR:
121B +1 246 EP7In_ISR :
121B +1 247 EP7Out_ISR:
121B +1 248 Not_Used: ; Should not get any of these
121B 32 +1 249 RETI
+1 250
121C +1 251 ClearINT2: ; Tell the hardware that we're done
121C E591 +1 252 MOV A, EXIF
121E C2E4 +1 253 CLR ACC.4 ; Clear the Interrupt 2 bit
1220 F591 +1 254 MOV EXIF, A
1222 22 +1 255 RET
+1 256
```

```
1223          +1 257 USBReset_ISR:                ; Bus has been Reset, move to DEFAULT state
1223 C200      +1 258          CLR          Configured
1225 9010AB    +1 259          MOV          DPTR, #(1000H OR LOW(USBIRQ))
1228          +1 260 ExitISR:                ; Common exit for all ISR's
          +1 261          ; On entry DPH = Interrupt ID, DPL = LOW(Interrupt Register)
1228 511C      +1 262          CALL         ClearINT2
122A 747F      +1 263          MOV          A, #7FH                ; EZ-USB I/O Register Page
122C C583      +1 264          XCH          A, DPH
122E F0        +1 265          MOVX         @DPTR, A                ; Clear source of interrupt
122F 32        +1 266          RETI
          +1 267
1230          +1 268 Suspend_ISR:                ; SIE detected an Idle bus
          +1 269          ; This routine assumes that MAIN has no context to save
1230 511C      +1 270          CALL         ClearINT2
1232 78AB      +1 271          MOV          R0, #LOW(USBIRQ)        ; PageReg = 7FH = HIGH(USBIRQ)
1234 7408      +1 272          MOV          A, #8
1236 F2        +1 273          MOVX         @R0, A                ; Clear the interrupt before suspending
1237 E4        +1 274          CLR          A
1238 7897      +1 275          MOV          R0, #LOW(PortB_Out)    ; PageReg = 7FH = HIGH(PortB_Out)
123A F2        +1 276          MOVX         @R0, A                ; Turn off LEDs
123B E587      +1 277          MOV          A, PCON
123D C2E0      +1 278          CLR          ACC.0
123F F587      +1 279          MOV          PCON, A                ; Turn off the oscillator
1241 00        +1 280          NOP
1242 00        +1 281 IDLE:                ; Enter IDLE mode
1243 00        +1 282          NOP
1244 32        +1 283          RETI
          +1 284
1245          +1 285 Wakeup_ISR:                ; Wake up from IDLE mode
1245 C2DC      +1 286          CLR          EICON.4                ; Clear the interrupt signal
          +1 287          ; This routine assumes that MAIN has no context to save
1247 78D6      +1 288          MOV          R0, #LOW(USBControl)
1249 E2        +1 289          MOVX         A, @R0                ; Get USB Control/Status register
124A 20E701    +1 290          JB          ACC.7, IODeviceRequest
124D          +1 291 PChostRequest:                ; The wakeup was caused by the PC Host
124E 32        +1 292          RETI
124E          +1 293 IODeviceRequest:                ; The IO device wants to wakeup the PC Host
          +1 294          ; NOTE that this capability is included in the Configuration Descriptor
124E 4581      +1 295          ORL          A, 10000001B
1250 F2        +1 296          MOVX         @R0, A                ; Clear Wakeup source and signal a RESUME
1251 557E      +1 297          ANL          A, 01111110B
1253 FA        +1 298          MOV          R2, A                ; Save ACC
1254 7F0F      +1 299          MOV          R7, #15
1256 7155      +1 300          CALL         Wait1msec                ; Signal a RESUME for 15ms
1258 EA        +1 301          MOV          A, R2
1259 F2        +1 302          MOVX         @R0, A                ; Remove RESUME signal
          +1 303          ; We're done, return to the Suspend_ISR
125A 32        +1 304          RETI
          +1 305
125B          +1 306 EP0In_ISR:                ; A prepared packet has been read by PC host
125B E546      +1 307          MOV          A, SaveLength                ; Do I have any more data to send?
125D 6008      +1 308          JZ          NoMoreToSend
125F 854483    +1 309          MOV          DPH, SaveDPH                ; Retrieve descriptor pointer
1262 854582    +1 310          MOV          DPL, SaveDPL
1265 51D0      +1 311          CALL         SendNextPieceOfDescriptor
1267          +1 312 NoMoreToSend:
1267 9001A9    +1 313          MOV          DPTR, #(100H OR LOW(IN07IRQ))
126A 80BC      +1 314          JMP          ExitISR
          +1 315
126C          +1 316 SOF_ISR:                ; A Start-Of-Frame packet has been received
          +1 317          ; CALL         ServiceTimerRoutine
          +1 318          ; This routine services the real time interrupt
          +1 319          ; It is also responsible for the "real world" buttons and lights
          +1 320          ;
126C          +1 321 ServiceTimerRoutine:
126C D5520F    +1 322          DJNZ         Msec_counter, Done                ; Only need to check every 4msec
```

```
126F 755204 +1 323 MOV Msec_counter, #4 ; Reinitialize
+1 324 ; LED task
1272 E551 +1 325 MOV A, LEDValue
1274 907F97 +1 326 MOV DPTR, #PortB_Out
1277 F0 +1 327 MOVX @DPTR, A ; Update the real world
+1 328 ;
+1 329 ; Create an Input Report from the Buttons value
+1 330 ; This will be continually overwritten while the PCHost is not polling for data
1278 +1 331 ReadButtons:
1278 907F99 +1 332 MOV DPTR, #PortA_Pins
127B E0 +1 333 MOVX A, @DPTR
127C 716C +1 334 CALL CreateInputReport
127E 9002AB +1 335 Done: MOV DPTR, #(200H OR LOW(USBRQ))
1281 80A5 +1 336 JMP ExitISR
+1 337
1283 +1 338 SUDAV_ISR: ; A Setup packet has been received
1283 754600 +1 339 MOV SaveLength, #0 ; Clear any pending transactions (if any)
1286 907FE8 +1 340 MOV DPTR, #SETUPDAT ; Copy packet to direct access memory
1289 7847 +1 341 MOV R0, #SetupData
128B 7F08 +1 342 MOV R7, #8
128D E0 +1 343 CopySD: MOVX A, @DPTR
128E F6 +1 344 MOV @R0, A
128F A3 +1 345 INC DPTR
1290 08 +1 346 INC R0
1291 DFFA +1 347 DJNZ R7, CopySD
1293 717A +1 348 CALL ServiceSetupPacket ; Handle the decode of the Setup packet
+1 349 ; if SetAddress { Update SIE address } // NOP on EZ-USB
+1 350 ; if STALL { Stall the endpoint }
+1 351 ; if SendData {
+1 352 ; if IsDescriptor { send DPTR->descriptor, A = length }
+1 353 ; else { send ReplyBuffer }
+1 354 ; }
1295 200126 +1 355 JB STALL, SendSTALL
1298 300216 +1 356 JNB SendData, HandShake
129B 200324 +1 357 JB IsDescriptor, LoadEP0
+1 358 ; Send data in ReplyBuffer
129E 907F01 +1 359 MOV DPTR, #EP0InBuffer+1
12A1 7842 +1 360 MOV R0, #ReplyBuffer+1
12A3 7F02 +1 361 MOV R7, #2 ; Copy the two byte buffer
12A5 E6 +1 362 CopyRB: MOV A, @R0
12A6 F0 +1 363 MOVX @DPTR, A
12A7 1582 +1 364 DEC DPL
12A9 18 +1 365 DEC R0
12AA DFF9 +1 366 DJNZ R7, CopyRB
12AC E6 +1 367 MOV A, @R0 ; Get BufferCount
12AD +1 368 SendEP0InBuffer:
12AD 907FB5 +1 369 MOV DPTR, #In0ByteCount
12B0 +1 370 StartXfer:
12B0 F0 +1 371 MOVX @DPTR, A ; This write initiates the transfer
12B1 +1 372 HandShake: ; Handshake with host
12B1 7F02 +1 373 MOV R7, #00000010b ; Set HSNACK to tell the SIE that we're done
12B3 +1 374 SetEP0Control:
12B3 907FB4 +1 375 MOV DPTR, #EP0Control
12B6 E0 +1 376 MOVX A, @DPTR
12B7 4F +1 377 ORL A, R7
12B8 F0 +1 378 MOVX @DPTR, A ; We're done
12B9 9001AB +1 379 MOV DPTR, #(100H OR LOW(USBRQ))
12BC 4128 +1 380 JMP ExitISR
12BE +1 381 SendSTALL: ; Invalid Request was received
12BE 7F03 +1 382 MOV R7, #00000011b ; Set EP0STALL and HSNACK
12C0 80F1 +1 383 JMP SetEP0Control
12C2 +1 384 LoadEP0: ; Send the data pointed to by DPTR
12C2 FF +1 385 MOV R7, A ; Save LENGTH
+1 386 ; Need to return the smaller of "Requested Length" and "Actual Length"
+1 387 ; If "Requested Length" > 255 then use "Actual Length"
+1 388 ; There are no descriptors > 255 in this example
```

```
12C3 E54E      +1 389          MOV     A, wLengthHigh
12C5 7008      +1 390          JNZ     UseActual
12C7 EF        +1 391          MOV     A, R7              ; Retrieve LENGTH
12C8 C3        +1 392          CLR     C
12C9 954D      +1 393          SUBB   A, wLengthLow
12CB E54D      +1 394          MOV     A, wLengthLow     ; This does not affect Carry
12CD 5001      +1 395          JNC     UseLengthLow
12CF EF        +1 396          UseActual:
12CF EF        +1 397          MOV     A, R7
12D0           +1 398          UseLengthLow:
12D0           +1 399          SendNextPieceOfDescriptor: ; DPTR -> Descriptor to be sent
12D0 FF        +1 400          MOV     R7, A             ; Save LENGTH again
12D1 754600    +1 401          MOV     SaveLength, #0    ; Default case, overwrite if necessary
+1 402          ; Do I have more than a single packet to send?
12D4 C3        +1 403          CLR     C
12D5 9440      +1 404          SUBB   A, #EP0Size
12D7 4015      +1 405          JC      SendPacket
+1 406          ; Need to send multiple packets.
+1 407          ; Calculate and save address of next packet, send next packet now
12D9 F546      +1 408          MOV     SaveLength, A     ; Send these next time
12DB 7F40      +1 409          MOV     R7, #EP0Size
12DD C083      +1 410          PUSH   DPH              ; Save current pointer
12DF C082      +1 411          PUSH   DPL
12E1 EF        +1 412          MOV     A, R7              ; Retrieve length
12E2 71BD      +1 413          CALL   BumpDPTR
12E4 858344    +1 414          MOV     SaveDPH, DPH
12E7 858245    +1 415          MOV     SaveDPL, DPL
12EA D082      +1 416          POP    DPL
12EC D083      +1 417          POP    DPH
12EE           +1 418          SendPacket:
12EE EF        +1 419          MOV     A, R7              ; Retrieve length
12EF FE        +1 420          MOV     R6, A             ; Save length in R6 for move
12F0 7800      +1 421          MOV     R0, #LOW(EP0InBuffer) ; PageReg = 7FH = HIGH(EP0InBuffer)
12F2 E0        +1 422          CopySTD:MOVX   A, @DPTR
12F3 F2        +1 423          MOVX   @R0, A
12F4 A3        +1 424          INC    DPTR
12F5 08        +1 425          INC    R0
12F6 DEFA      +1 426          DJNZ   R6, CopySTD
12F8 EF        +1 427          MOV     A, R7              ; Retrieve LENGTH
12F9 80B2      +1 428          JMP     SendEP0InBuffer
+1 429
12FB           +1 430          GetOutputReport:         ; Wait for this, it's next on USB
12FB 907FC5    +1 431          MOV     DPTR, #Out0ByteCount ; Enable EP0OutBuffer to receive data
12FE F0        +1 432          MOVX   @DPTR, A           ; Any value will do
12FF 907FB4    +1 433          MOV     DPTR, #EP0Control   ; Wait for valid data in EP0OutBuffer
1302 E0        +1 434          Wait40:MOVX   A, @DPTR
1303 5408      +1 435          ANL    A, #00001000b       ; Check OUTBSY
1305 70FB      +1 436          JNZ    Wait40
1307 22        +1 437          RET
+1 438
+1 439
+1 440
+1 441
+1 442          ;$include (EZMain.A51)
+1 443          ; This module initializes the microcontroller then executes MAIN forever
+1 444          ; It is hardware dependant
+1 445
1308           +1 446          Reset:
1308 7581DF    +1 447          MOV     SP, #0DFH          ; Initialize the Stack
130B 75927F    +1 448          MOV     PageReg, #7FH     ; Allows MOVX Ri to access EZ-USB memory
+1 449
130E 78D6      +1 450          MOV     R0, #Low(USBControl) ; Simulate a disconnect
1310 E2        +1 451          MOVX   A, @R0
1311 54F3      +1 452          ANL    A, #11110011b     ; Clear DISCON, DISCOE
1313 F2        +1 453          MOVX   @R0, A
1314 7153      +1 454          CALL   Wait200msec       ; Give the host time to react
```

```
1316 E2      +1  455      MOVX    A, @R0                ; Reconnect with this new identity
1317 4406     +1  456      ORL     A, #00000110b       ; Set DISCOE to enable pullup resistor
1319 F2      +1  457      MOVX    @R0, A              ; Set RENUM so that 8051 handles USB requests
131A E4      +1  458      CLR     A
131B F520     +1  459      MOV     FLAGS, A            ; Start in Default state
131D         +1  460      TurnOffLEds:
131D F551     +1  461      MOV     LEDValue, A
131F F54F     +1  462      MOV     Old_Buttons, A
1321 04      +1  463      INC     A                    ; = 1
1322 F550     +1  464      MOV     LEDstrobe, A
1324         +1  465      Initialize4msecCounter:
1324 F552     +1  466      MOV     Msec_counter, A
1326         +1  467      InitializeIOSystem:       ; Setup for Simmbus A=input, B=output
                                ; C=External RD#,WR#,TD0,TR0
                                ; PageReg = 7F = HIGH(PortA_Config)
1326 7893     +1  469      MOV     R0, #LOW(PortA_Config)
1328 799C     +1  470      MOV     R1, #LOW(PortA_OE)
132A E4      +1  471      CLR     A
132B F2      +1  472      MOVX    @R0, A              ; No alternate functions
132C F3      +1  473      MOVX    @R1, A              ; Enable PortA for Input
132D 08      +1  474      INC     R0                  ; Point to PortB_Config
132E 09      +1  475      INC     R1                  ; Point to PortB_OE
132F F2      +1  476      MOVX    @R0, A              ; No alternate functions
1330 F4      +1  477      CPL     A                    ; = OFFH
1331 F3      +1  478      MOVX    @R1, A              ; Enable PortB for Output
1332 08      +1  479      INC     R0                  ; Point to PortC_Config
1333 09      +1  480      INC     R1                  ; Point to PortC_OE
1334 74C3     +1  481      MOV     A, #11000011b
1336 F2      +1  482      MOVX    @R0, A              ; Alternate functions on [7,6,1,0]
1337 74C2     +1  483      MOV     A, #11000010b
1339 F3      +1  484      MOVX    @R1, A              ; Most alternate functions are outputs
133A         +1  485      InitializeInterruptSystem: ; First initialize the USB level
133A 7401     +1  486      MOV     A, #00000001b
133C 78AC     +1  487      MOV     R0, #LOW(IN07IEN)
133E F2      +1  488      MOVX    @R0, A              ; Enable interrupts from EP0IN only
133F 08      +1  489      INC     R0
1340 E4      +1  490      CLR     A
1341 F2      +1  491      MOVX    @R0, A              ; Disable interrupts from OUT Endpoints 0-7
1342 08      +1  492      INC     R0
1343 7413     +1  493      MOV     A, #00010011b
                                ; Enable USBReset, (Resume, Suspend,) SOF and SUDAV INTs
1345 F2      +1  495      MOVX    @R0, A
1346 08      +1  496      INC     R0
1347 7401     +1  497      MOV     A, #00000001b
1349 F2      +1  498      MOVX    @R0, A              ; Enable Auto Vectoring for USB interrupts
                                ; Now enable the main level
134A 75E801   +1  500      MOV     EIE, #00000001b     ; Enable INT2 = USB Interrupt (only)
134D 75A890   +1  501      MOV     EI, #10010000b     ; Enable interrupt subsystem (and Ser0 for
dScope)
                                +1  502
                                +1  503      ; Initialization Complete.
                                +1  504      ;
1350         +1  505      MAIN:
1350 00      +1  506      NOP                        ; Not much of a main loop for this example
1351 80FD     +1  507      JMP     MAIN                ; All actions are initiated by interrupts
                                +1  508      ; We are a slave, we wait to be told what to do
                                +1  509
1353         +1  510      Wait200msec:
1353 7FC8     +1  511      MOV     R7, #200
1355         +1  512      Wait1msec:                ; A delay loop
                                ; Select primary DPTR
1355 758600   +1  513      MOV     DPS, #0
1358 90FB50   +1  514      MOV     DPTR, #-1200
135B A3      +1  515      More:    INC     DPTR        ; 3 cycles
135C E582     +1  516      MOV     A, DPL             ; + 2
135E 4583     +1  517      ORL     A, DPH             ; + 2
1360 70F9     +1  518      JNZ    More                ; + 3 = 10 cycles x 1200 = 1msec
1362 DFF1     +1  519      DJNZ   R7, Wait1msec
1364 22      +1  520      RET
```

```
+1 521
1365 +1 522 ProcessOutputReport: ; A Report has just been received
+1 523 ; The report is only one byte long in this first example
+1 524 ; It contains a new value for the LEDs
1365 907EC0 +1 525 MOV DPTR, #EP0OutBuffer ; Point to the Report
1368 E0 +1 526 MOVX A, @DPTR ; Get the Data
1369 F551 +1 527 MOV LEDValue, A ; Update the local variable
136B 22 +1 528 RET
+1 529
136C +1 530 CreateInputReport: ; Called from TIMER which detected the need
+1 531 ; The report is only one byte long in this first example
+1 532 ; It contains a new value for the Buttons
136C 30000A +1 533 JNB Configured, NoReport ; Must be Configured to create reports
136F 907E80 +1 534 MOV DPTR, #EP1InBuffer ; Point to the buffer
1372 F0 +1 535 MOVX @DPTR, A ; Update the Report
1373 907FB7 +1 536 MOV DPTR, #IN1ByteCount
1376 7401 +1 537 MOV A, #1
1378 F0 +1 538 MOVX @DPTR, A ; Endpoint 1 now 'armed', next IN will get data
1379 +1 539 NoReport:
1379 22 +1 540 RET
+1 541
+1 542
+1 543 ;$include (Decode.A51)
+1 544 ; This module is common to all of the examples.
+1 545 ; It decodes the USB Setup Packets and generates appropriate responses.
+1 546 ; Interpretation of Reports is handled by MAIN
+1 547 ;
---- +1 548 CSEG
137A +1 549 ServiceSetupPacket:
137A E547 +1 550 MOV A, RequestType
137C A2E7 +1 551 MOV C, ACC.7 ; Bit 7 = 1 means IO device needs to send data
to P
C Host
137E 9202 +1 552 MOV SendData, C
1380 545C +1 553 ANL A, #01011100b ; IF RequestType[6.4.3.2] = 1 THEN goto
BadRequest
1382 7035 +1 554 JNZ BadRequest
1384 E547 +1 555 MOV A, RequestType ; IF RequestType[1&0] = 1 THEN goto BadRequest
1386 A2E0 +1 556 MOV C, ACC.0
1388 82E1 +1 557 ANL C, ACC.1
138A 402D +1 558 JC BadRequest
138C 30E502 +1 559 JNB ACC.5, NotB5 ; IF RequestType[5] = 1 THEN RequestType[1,0] =
[1,
1]
138F 7403 +1 560 MOV A, #00000011b
1391 5403 +1 561 NotB5: ANL A, #00000011b ; Set CommandIndex[5,4] = RequestType[1,0]
1393 C4 +1 562 SWAP A
1394 FF +1 563 MOV R7, A ; Save HI nibble of CommandIndex
+1 564 ; Set CommandIndex[3,0] = Request[3,0]
1395 E548 +1 565 MOV A, Request
1397 54F0 +1 566 ANL A, #11110000b ; Check if Request > 15
1399 701E +1 567 JNZ BadRequest
139B E548 +1 568 MOV A, Request
139D 540F +1 569 ANL A, #00001111b ; Only 13 are defined today, handle in table
139F 4F +1 570 ORL A, R7
+1 571 ; CALL CorrectSubroutine ; goto CommandTable(CommandIndex)
13A0 +1 572 CorrectSubroutine: ; Jump to the subroutine that DPTR is pointing
to
13A0 754001 +1 573 MOV ReplyCount, #1 ; Set up a default reply
13A3 754100 +1 574 MOV ReplyBuffer, #0
13A6 754200 +1 575 MOV ReplyBuffer+1, #0
13A9 C204 +1 576 CLR SetAddress ; Clear all flags
13AB C201 +1 577 CLR STALL
13AD C203 +1 578 CLR IsDescriptor
13AF 9013C6 +1 579 MOV DPTR, #CommandTable
13B2 71BD +1 580 CALL BumpDPTR ; Point to entry
13B4 E0 +1 581 MOVX A, @DPTR ; Get the offset
13B5 901406 +1 582 MOV DPTR, #Subroutines
13B8 73 +1 583 JMP @A+DPTR ; Go to the correct Subroutine
+1 584
```

```

13B9      +1  585      BadRequest:                ; Decoded a Bad Request, STALL the Endpoint
13B9 D201  +1  586          SETB      STALL
13BB 22    +1  587          RET
13BC      +1  588          ; Support routines
13BC E0    +1  589      NextDPTR:                ; Returns (DPTR + byte DPTR is pointing to)
13BC      +1  590          MOVX      A, @DPTR
13BD      +1  591      BumpDPTR:                ; Returns (DPTR + ACC)
13BD 2582  +1  592          ADD      A, DPL
13BF F582  +1  593          MOV      DPL, A
13C1 5002  +1  594          JNC      Skip
13C3 0583  +1  595          INC      DPH                ; Need 16 bit arithmetic here
13C5 22    +1  596      Skip:      RET
13C5      +1  597
13C5      +1  598      ; Since the table only contains byte offsets, it is important that all these routines
are
13C5      +1  599      ; within one page (100H) of Subroutines
13C5      +1  600      ; V3.0 - CommandTable moved outside of this one page limited space
13C6      +1  601      CommandTable:
13C6      +1  602      ; First 16 commands are for the Device
13C6 18    +1  603          DB LOW(Device_Get_Status - Subroutines)
13C7 00    +1  604          DB LOW(Device_Clear_Feature - Subroutines)
13C8 00    +1  605          DB LOW(Invalid - Subroutines)
13C9 00    +1  606          DB LOW(Device_Set_Feature - Subroutines)
13CA 00    +1  607          DB LOW(Invalid - Subroutines)
13CB 03    +1  608          DB LOW(Set_Address - Subroutines)
13CC 32    +1  609          DB LOW(Get_Descriptor - Subroutines)
13CD 00    +1  610          DB LOW(Set_Descriptor - Subroutines)
13CE 14    +1  611          DB LOW(Get_Configuration - Subroutines)
13CF 20    +1  612          DB LOW(Set_Configuration - Subroutines)
13D0 00    +1  613          DB LOW(Invalid - Subroutines)
13D1 00    +1  614          DB LOW(Invalid - Subroutines)
13D2 00    +1  615          DB LOW(Invalid - Subroutines)
13D3 00    +1  616          DB LOW(Invalid - Subroutines)
13D4 00    +1  617          DB LOW(Invalid - Subroutines)
13D5 00    +1  618          DB LOW(Invalid - Subroutines)
13D5      +1  619      ; Next 16 commands are for the Interface
13D6 1C    +1  620          DB LOW(Interface_Get_Status - Subroutines)
13D7 00    +1  621          DB LOW(Interface_Clear_Feature - Subroutines)
13D8 00    +1  622          DB LOW(Invalid - Subroutines)
13D9 00    +1  623          DB LOW(Interface_Set_Feature - Subroutines)
13DA 00    +1  624          DB LOW(Invalid - Subroutines)
13DB 00    +1  625          DB LOW(Invalid - Subroutines)
13DC 59    +1  626          DB LOW(Get_Class_Descriptor - Subroutines)
13DD 00    +1  627          DB LOW(Set_Class_Descriptor - Subroutines)
13DE 00    +1  628          DB LOW(Invalid - Subroutines)
13DF 00    +1  629          DB LOW(Invalid - Subroutines)
13E0 00    +1  630          DB LOW(Get_Interface - Subroutines)
13E1 00    +1  631          DB LOW(Set_Interface - Subroutines)
13E2 00    +1  632          DB LOW(Invalid - Subroutines)
13E3 00    +1  633          DB LOW(Invalid - Subroutines)
13E4 00    +1  634          DB LOW(Invalid - Subroutines)
13E5 00    +1  635          DB LOW(Invalid - Subroutines)
13E5      +1  636      ; Next 16 commands are for the Endpoint
13E6 1C    +1  637          DB LOW(Endpoint_Get_Status - Subroutines)
13E7 00    +1  638          DB LOW(Endpoint_Clear_Feature - Subroutines)
13E8 00    +1  639          DB LOW(Invalid - Subroutines)
13E9 00    +1  640          DB LOW(Endpoint_Set_Feature - Subroutines)
13EA 00    +1  641          DB LOW(Invalid - Subroutines)
13EB 00    +1  642          DB LOW(Invalid - Subroutines)
13EC 00    +1  643          DB LOW(Invalid - Subroutines)
13ED 00    +1  644          DB LOW(Invalid - Subroutines)
13EE 00    +1  645          DB LOW(Invalid - Subroutines)
13EF 00    +1  646          DB LOW(Invalid - Subroutines)
13F0 00    +1  647          DB LOW(Invalid - Subroutines)
13F1 00    +1  648          DB LOW(Invalid - Subroutines)
13F2 00    +1  649          DB LOW(Endpoint_Sync_Frame - Subroutines)
13F3 00    +1  650          DB LOW(Invalid - Subroutines)

```

```
13F4 00      +1  651          DB LOW(Invalid - Subroutines)
13F5 00      +1  652          DB LOW(Invalid - Subroutines)
              +1  653      ; Next 16 commands are Class Requests
13F6 00      +1  654          DB LOW(Invalid - Subroutines)
13F7 0D      +1  655          DB LOW(Get_Report - Subroutines)
13F8 00      +1  656          DB LOW(Get_Idle - Subroutines)
13F9 00      +1  657          DB LOW(Get_Protocol - Subroutines)
13FA 00      +1  658          DB LOW(Invalid - Subroutines)
13FB 00      +1  659          DB LOW(Invalid - Subroutines)
13FC 00      +1  660          DB LOW(Invalid - Subroutines)
13FD 00      +1  661          DB LOW(Invalid - Subroutines)
13FE 00      +1  662          DB LOW(Invalid - Subroutines)
13FF 06      +1  663          DB LOW(Set_Report - Subroutines)
1400 00      +1  664          DB LOW(Set_Idle - Subroutines)
1401 00      +1  665          DB LOW(Set_Protocol - Subroutines)
1402 00      +1  666          DB LOW(Invalid - Subroutines)
1403 00      +1  667          DB LOW(Invalid - Subroutines)
1404 00      +1  668          DB LOW(Invalid - Subroutines)
1405 00      +1  669          DB LOW(Invalid - Subroutines)
              +1  670
1406         +1  671      Subroutines:
              +1  672      ;
              +1  673      ; Many requests are INVALID for this example
1406         +1  674      Get_Protocol:          ; We are not a Boot device
1406         +1  675      Set_Protocol:          ; We are not a Boot device
1406         +1  676      Set_Descriptor:        ; Our Descriptors are static
1406         +1  677      Set_Class_Descriptor:  ; Our Descriptors are static
1406         +1  678      Set_Interface:        ; We only have one Interface
1406         +1  679      Get_Interface:        ; We do not have an Alternate setting
1406         +1  680      Set_Idle:            ; V3.0 Optional command, not supported
1406         +1  681      Get_Idle:            ; V3.0 Optional command, not supported
1406         +1  682      Device_Set_Feature:   ; We have no features that can be set or cleared
1406         +1  683      Interface_Set_Feature: ; We have no features that can be set or cleared
1406         +1  684      Endpoint_Set_Feature: ; We have no features that can be set or cleared
1406         +1  685      Endpoint_Clear_Feature: ; V3.0 We have no features that can be set or cleared
1406         +1  686      Device_Clear_Feature: ; We have no features that can be set or cleared
1406         +1  687      Interface_Clear_Feature: ; We have no features that can be set or cleared
1406         +1  688      Endpoint_Sync_Frame:  ; We are not an Isonchronous device
              +1  689
1406         +1  690      Invalid:                ; Invalid Request made, STALL the Endpoint
1406 D201     +1  691          SETB      STALL
1408 22      +1  692      Reply:  RET
              +1  693
1409         +1  694      Set_Address:          ; Set the address that the SIE will respond to
1409 D204     +1  695          SETB      SetAddress
140B 22      +1  696          RET
              +1  697
140C         +1  698      Set_Report:          ; Host wants to sent us a Report.
              +1  699      ; The ONLY case in this example where host sends data to us
140C 3000F7   +1  700          JNB      Configured, Invalid ; Need to be Configured to do this command
140F 51FB     +1  701          CALL     GetOutputReport ; Handled in EZUSB.A51
1411 6165     +1  702          JMP      ProcessOutputReport ; RETurn via this subroutine
1413         +1  703      Get_Report:          ; Host wants a Report
1413 3000F0   +1  704          JNB      Configured, Invalid ; Need to be Configured to do this command
1416 754142   +1  705          MOV      ReplyBuffer, #42H ; Reply with a recognizable (arbitrary) value
1419 22      +1  706          RET
141A         +1  707      Get_Configuration:    ; Respond with CurrentConfiguration
141A 854341   +1  708          MOV      ReplyBuffer, CurrentConfiguration
141D 22      +1  709          RET
141E         +1  710      Device_Get_Status:      ; Only two bits of Device Status are defined
141E 754101   +1  711          MOV      ReplyBuffer, #1 ; Bit 1=Remote Wakeup(=0), Bit 0=Self
Powered(=1)
1421 22      +1  712          RET
1422         +1  713      Interface_Get_Status:    ; Interface Status is currently defined as 0
1422         +1  714      Endpoint_Get_Status:
1422 754002   +1  715          MOV      ReplyCount, #2 ; Need a two byte 0 response
1425 22      +1  716          RET
```

```
1426          +1 717      Set_Configuration:                ; Valid values are 0 and 1
1426 E549     +1 718          MOV      A, wValueLow
1428 6009     +1 719          JZ       Deconfigured
142A 14       +1 720          DEC      A
142B 70D9     +1 721          JNZ     Invalid
142D D200     +1 722          SETB   Configured
142F 754301   +1 723          MOV     CurrentConfiguration, #1
1432 22       +1 724          RET
1433          +1 725      Deconfigured:
1433 C200     +1 726          CLR     Configured
1435 F543     +1 727          MOV     CurrentConfiguration, A
1437 22       +1 728          RET
1438          +1 729      Get_Descriptor:                ; Host wants to know who/what we are
1438 D203     +1 730          SETB   IsDescriptor
143A E54A     +1 731          MOV     A, wValueHigh
143C 14       +1 732          DEC     A                ; Valid Values are 1, 2 and 3
143D 901478   +1 733          MOV     DPTR, #DeviceDescriptor
1440 6031     +1 734          JZ     ReturnLength
1442 14       +1 735          DEC     A
1443 90148A   +1 736          MOV     DPTR, #ConfigurationDescriptor
1446 7003     +1 737          JNZ     TryString
1448 7422     +1 738          MOV     A, #ConfigLength
144A 22       +1 739          RET
144B          +1 740      TryString:
144B 14       +1 741          DEC     A
144C 70B8     +1 742          JNZ     Invalid
          +1 743      ; Request is for a String Descriptor
144E 9014C8   +1 744          MOV     DPTR, #String0                ; Point to String 0
1451 E549     +1 745          MOV     A, wValueLow                ; Get String Index
1453          +1 746      NextString:
1453 601E     +1 747          JZ     ReturnLength
1455 FF       +1 748          MOV     R7, A                ; Save String Index
1456 71BC     +1 749          CALL   NextDPTR
1458 E0       +1 750          MOVX   A, @DPTR                ; Get the String Length (= 0 means we're at
Backsto
          p)
1459 60AB     +1 751          JZ     Invalid                ; Asked for a string I don't have
145B EF       +1 752          MOV     A, R7
145C 14       +1 753          DEC     A
145D 80F4     +1 754          JMP     NextString                ; Check if we are there yet
145F          +1 755      Get_Class_Descriptor:            ; Valid values are 21H, 22H, 23H for Class
Request
145F D203     +1 756          SETB   IsDescriptor
1461 E54A     +1 757          MOV     A, wValueHigh
1463 C3       +1 758          CLR     C
1464 9421     +1 759          SUBB   A, #21H
1466 90149C   +1 760          MOV     DPTR, #HIDDescriptor
1469 6008     +1 761          JZ     ReturnLength
146B 14       +1 762          DEC     A
146C 9014AC   +1 763          MOV     DPTR, #ReportDescriptor
146F 6004     +1 764          JZ     ReturnRDlength
          +1 765      ; DEC     A                ; This example does not use Physical
Descriptors
          +1 766      ; JZ     Send_Physical_Descriptor
1471 8093     +1 767          JMP     Invalid
          +1 768      ;
1473          +1 769      ReturnLength:
1473 E0       +1 770          MOVX   A, @DPTR                ; Get Descriptor Length (first byte)
1474 22       +1 771          RET
1475          +1 772      ReturnRDlength:                ; Report Descriptor is different format
1475 741C     +1 773          MOV     A, #ReportLength
1477 22       +1 774          RET
          +1 775      ; Error check: this MUST be on within a page of Subroutines
          +1 776      WithinSamePage EQU $ - Subroutines
          +1 777      ;
          +1 778
          +1 779
          +1 780      ;$include (DTables.A51)
          +1 781      ; This module declares the descriptors
```

```

+1 782 ;
+1 783 ; This example has one Device Descriptor with:
+1 784 ;     One Configuration - single IN port and single OUT port
+1 785 ;     One Interface - there is only one method of accessing the ports
+1 786 ;     One HID Descriptor - to make PC host software simpler
+1 787 ;     One Endpoint Descriptor - for HID Input Reports
+1 788 ;     One Report Descriptor - one byte IN and one byte OUT reports
+1 789 ;     Multiple Sting Descriptors - to aid the user
+1 790 ;
---- +1 791 ; CSEG
1478 +1 792 DeviceDescriptor:
1478 1201 +1 793     DB 18, 1 ; Length, Type
147A 1001 +1 794     DB 10H, 1 ; USB Rev 1.1 (=0110H, low=10H, High=01H)
147C 000000 +1 795     DB 0, 0, 0 ; Class, Subclass and Protocol
147F 40 +1 796     DB EPOSize
1480 42420142 +1 797     DB 42H, 42H, 1, 42H, 0, 1; Vendor ID, Product ID and Version
1484 0001
1486 010200 +1 798     DB 1, 2, 0 ; Manufacturer, Product & Serial# Names
1489 01 +1 799     DB 1 ; #Configs
148A +1 800 ConfigurationDescriptor:
148A 0902 +1 801     DB 9, 2 ; Length, Type
148C 2200 +1 802     DB LOW(ConfigLength), HIGH(ConfigLength)
148E 010100 +1 803     DB 1, 1, 0 ; #Interfaces, Configuration#, Config. Name
1491 A0 +1 804     DB 10100000b ; Attributes = Bus Powered
1492 FA +1 805     DB 250 ; Max. Power is 250x2 = 500mA
1493 +1 806 InterfaceDescriptor:
1493 0904 +1 807     DB 9, 4 ; Length, Type
1495 000001 +1 808     DB 0, 0, 1 ; No alternate setting, HID uses EP1
1498 03 +1 809     DB 3 ; Class = Human Interface Device
1499 0000 +1 810     DB 0, 0 ; Subclass and Protocol
149B 00 +1 811     DB 0 ; Interface Name
149C +1 812 HIDDescriptor:
149C 0921 +1 813     DB 9, 21H ; Length, Type
149E 0001 +1 814     DB 0, 1 ; HID Class Specification compliance
14A0 00 +1 815     DB 0 ; Country localization (=none)
14A1 01 +1 816     DB 1 ; Number of descriptors to follow
14A2 22 +1 817     DB 22H ; And it's a Report descriptor
14A3 1C00 +1 818     DB LOW(ReportLength), HIGH(ReportLength)
14A5 +1 819 EndpointDescriptor:
14A5 0705 +1 820     DB 7, 5 ; Length, Type
14A7 81 +1 821     DB 10000001b ; Address = IN 1
14A8 03 +1 822     DB 00000011b ; Interrupt
14A9 4000 +1 823     DB EPOSize, 0 ; Maximum packet size (this example only uses 1)
14AB 64 +1 824     DB 100 ; Poll every 0.1 seconds
      0022 +1 825     ConfigLength EQU $ - ConfigurationDescriptor
+1 826
14AC +1 827 ReportDescriptor: ; Generated with HID Tool, copied to here
14AC 0600FF +1 828     DB 6, 0, 0FFH ; Usage_Page (Vendor Defined)
14AF 0901 +1 829     DB 9, 1 ; Usage (I/O Device)
14B1 A101 +1 830     DB 0A1H, 1 ; Collection (Application)
14B3 1901 +1 831     DB 19H, 1 ; Usage_Minimum (Button 1)
14B5 2908 +1 832     DB 29H, 8 ; Usage_Maximum (Button 8)
14B7 1500 +1 833     DB 15H, 0 ; Logical_Minimum (0)
14B9 2501 +1 834     DB 25H, 1 ; Logical_Maximum (1)
14BB 7501 +1 835     DB 75H, 1 ; Report_Size (1)
14BD 9508 +1 836     DB 95H, 8 ; Report_Count (8)
14BF 8102 +1 837     DB 81H, 2 ; Input (Data,Var,Abs)
14C1 1901 +1 838     DB 19H, 1 ; Usage_Minimum (Led 1)
14C3 2908 +1 839     DB 29H, 8 ; Usage_Maximum (Led 8)
14C5 9102 +1 840     DB 91H, 2 ; Output (Data,Var,Abs)
14C7 C0 +1 841     DB 0C0H ; End_Collection
      001C +1 842     ReportLength EQU $-ReportDescriptor
+1 843
14C8 +1 844 String0: ; Declare the UNICODE strings
14C8 04030904 +1 845     DB 4, 3, 9, 4 ; Only English language strings supported
14CC +1 846 String1: ; Manufacturer

```

```

14CC 2C03      +1  847          DB      (String2-String1),3 ; Length, Type
14CE 55005300 +1  848          DB      "U",0,"S",0,"B",0," ",0,"D",0,"e",0,"s",0,"i",0,"g",0,"n",0," ",0
14D2 42002000
14D6 44006500
14DA 73006900
14DE 67006E00
14E2 2000
14E4 42007900 +1  849          DB      "B",0,"y",0," ",0,"E",0,"x",0,"a",0,"m",0,"p",0,"l",0,"e",0
14E8 20004500
14EC 78006100
14F0 6D007000
14F4 6C006500
14F8           +1  850      String2:          ; Product Name
14F8 2203      +1  851          DB      (EndOfDescriptors-String2),3
14FA 42007500 +1  852          DB      "B",0,"u",0,"t",0,"t",0,"o",0,"n",0,"s",0," ",0
14FE 74007400
1502 6F006E00
1506 73002000
150A 26002000 +1  853          DB      "&",0," ",0,"L",0,"i",0,"g",0,"h",0,"t",0,"s",0
150E 4C006900
1512 67006800
1516 74007300
151A           +1  854      EndOfDescriptors:
151A 00        +1  855          DB      0          ; Backstop for String Descriptors
+1  856
+1  857
+1  858
+1  859
+1  860      ;$include (ProgEE.A51)
+1  861      ; This module copies the currently active program into the EEPROM of a USBSIMM
+1  862      ; This will enable the EZ-USB to "boot from EEPROM"
+1  863      ;
+1  864      ; It installs itself around the current program which will be data for the EEPROM
+1  865      ; It then sets a new start address so that it is the active program
+1  866      ; The interrupt vector table must be hand coded
+1  867      ; This module must be included LAST in a build
+1  868      ;
+1  869      ; A 24LC64 EEPROM is assumed (this 8K device is standard on USBSIMM)
+1  870      ; 24LC64 supports sequential reads up to 8K and sequential page writes of 32bytes
+1  871      ;
8000           +1  872      OldEEPROMdata EQU      8000H          ; Save area for current EEPROM contents
A000           +1  873      NewEEPROMdata EQU      0A000H        ; Save area for new EEPROM contents
+1  874
0080           +1  875      StartBit      DATA    10000000b
0040           +1  876      StopBit       DATA    01000000b
0020           +1  877      LastReadBit   DATA    00100000b
00A2           +1  878      EEPROMaddress DATA    10100010b
+1  879
031B           +1  880      ProgramLength EQU      $-USB_ISR
+1  881
151B           +1  882      EEPROMfooter:
151B 0003      +1  883          DB      0, 3          ; Length = 3
151D 0000      +1  884          DB      0, 0          ; Load address = 0
151F 021308    +1  885          LJMP     Reset        ; Start of MAIN program
1522 0003      +1  886          DB      0, 3          ; Length = 3
1524 0043      +1  887          DB      0, 43H         ; Load address = 43H = USB Vector
1526 021200    +1  888          LJMP     USB_ISR
1529 8001      +1  889          DB      80H, 1         ; Length = 1 (Special end marker)
152B 7F92      +1  890          DB      7FH, 92H        ; CPU_Control Register
152D 00        +1  891          DB      0          ; Brings 8051 out of RESET
152E           +1  892      EEPROMend     EQU      $
+1  893      ; End of data tables, program starts here
+1  894
152E           +1  895      SendWriteCommand:
152E 7AA2      +1  896          MOV      R2, #EEPROMaddress ;
1530           +1  897      SendCommand:    ; Send the first byte to I2C bus

```

```
1530 B14B      +1  898          ACALL   Wait4Stop
1532 7480      +1  899          MOV    A, #StartBit
1534 F2        +1  900          MOVX   @R0, A                ; I2C engine now primed
1535          +1  901      SendDataByte:
1535 EA        +1  902          MOV    A, R2                ; Get data and . . .
1536 F3        +1  903      SDB2:  MOVX   @R1, A                ; . . . send using I2C engine
1537          +1  904      Wait4Done:
1537 E2        +1  905          MOVX   A, @R0
1538 30E0FC    +1  906          JNB   ACC.0, Wait4Done      ; Wait for the data to be transmitted
153B 22        +1  907          RET
153C          +1  908
153C          +1  909      ReceiveDataByte:          ; Read from I2C
153C E3        +1  910          MOVX   A, @R1
153D F0        +1  911          MOVX   @DPTR, A
153E 80F7      +1  912          JMP    Wait4Done
153E          +1  913
1540          +1  914      ResetEEPROMAddress:
1540 B12E      +1  915          CALL   SendWriteCommand
1542 7A00      +1  916          MOV    R2, #0                ; Address inside 24LC64
1544 B135      +1  917          CALL   SendDataByte        ; HIGH byte = 0
1546 B135      +1  918          CALL   SendDataByte        ; LOW byte = 0
1546          +1  919          ; Fall into ...
1548          +1  920      SetStopBitandWait:
1548 7440      +1  921          MOV    A, #StopBit
154A F2        +1  922          MOVX   @R0, A
154B          +1  923      Wait4Stop:                ; Make sure the STOP from a previous
154B E2        +1  924          MOVX   A, @R0                ; I2C transaction has completed
154C 20E6FC    +1  925          JB    ACC.6, Wait4Stop
154F 22        +1  926          RET
154F          +1  927
154F          +1  928
1550          +1  929      ReadEEPROM:
1550          +1  930      ; Note: 24LC64 will auto-increment read address
1550 B140      +1  931          CALL   ResetEEPROMAddress
1552 7AA3      +1  932          MOV    R2, #EEPROMAddress OR 1 ; Set 'read'
1554 B130      +1  933          CALL   SendCommand
1556 E3        +1  934          MOVX   A, @R1                ; Trigger a read
1557 B137      +1  935          CALL   Wait4Done
1557          +1  936          ; Setup to read next 8K-2 bytes
1559 7FFD      +1  937          MOV    R7, #0FDH
155B 7E04      +1  938          MOV    R6, #4
155D          +1  939      ReadLoop:
155D B13C      +1  940          CALL   ReceiveDataByte
155F A3        +1  941          INC    DPTR                ; Bump Save Pointer
1560 DFFB      +1  942          DJNZ  R7, ReadLoop
1562 DEF9      +1  943          DJNZ  R6, ReadLoop
1562          +1  944          ; Now read the last two bytes. Need to set LastRd to stop reading the EEPROM
1564 7420      +1  945          MOV    A, #LastReadBit
1566 F2        +1  946          MOVX   @R0, A
1567 B13C      +1  947          CALL   ReceiveDataByte        ; Get penultimate byte
1569 A3        +1  948          INC    DPTR                ; Bump Save Pointer
156A E3        +1  949          MOVX   A, @R1                ; Get last byte
156B F0        +1  950          MOVX   @DPTR, A            ; And save it
156C 7440      +1  951          MOV    A, #StopBit
156E F2        +1  952          MOVX   @R0, A
156F 22        +1  953          RET
156F          +1  954
1570          +1  955      WriteEEPROM:
1570          +1  956      ; Note: 24LC64 writes pages in blocks of 32 bytes
1570 7D00      +1  957          MOV    R5, #0                ; EEPROM address HIGH
1572 7E00      +1  958          MOV    R6, #0                ; EEPROM address LOW
1572          +1  959          ; Blockcount is in R7
1574          +1  960      WriteLoop:
1574 7C20      +1  961          MOV    R4, #32                ; Blocksize
1576 B12E      +1  962          CALL   SendWriteCommand
1578 ED        +1  963          MOV    A, R5                ; Set HIGH current address inside 24LC64
```

```
1579 B136      +1  964      CALL    SDB2                ; Bypass R2 is SendDataByte
157B EE        +1  965      MOV     A, R6                ; Set LOW current address inside 24LC64
157C B136      +1  966      CALL    SDB2                ; Bypass R2 is SendDataByte
157E           +1  967      NextDataByte:
157E E0        +1  968      MOVX   A, @DPTR              ; Get data to be written
157F B136      +1  969      CALL    SDB2                ; Bypass R2 is SendDataByte
1581 A3        +1  970      INC    DPTR
1582 DCFA      +1  971      DJNZ   R4, NextDataByte
1584 B148      +1  972      CALL    SetStopBitandWait    ; New block sent to EEPROM
+1  973      ; Wait for the EEPROM to complete the write, this is about 4msec
+1  974      ; Send a Write Command and check ACK, ACK will be true when EEPROM is ready
1586 B12E      +1  975      Wait:   CALL    SendWriteCommand
1588 B148      +1  976      CALL    SetStopBitandWait
158A E2        +1  977      MOVX   A, @R0
158B 30E1F8    +1  978      JNB    ACC.1, Wait          ; Wait for ACK=1
158E 7420      +1  979      MOV    A, #32                ; Update EEPROM address pointer
1590 2E        +1  980      ADD    A, R6
1591 FE        +1  981      MOV    R6, A
1592 7400      +1  982      MOV    A, #0
1594 3D        +1  983      ADDC   A, R5
1595 FD        +1  984      MOV    R5, A
1596 DFDC      +1  985      DJNZ   R7, WriteLoop
1598 22        +1  986      RET
+1  987
+1  988
1599           +1  989      ProgEEPROM:
1599 7581DF     +1  990      MOV    SP, #0DFH            ; Need a small stack
159C 75927F    +1  991      MOV    PageReg, #7FH        ; Point to EZ-USB internal registers
+1  992      ; Now initialize I2C
159F 78A5     +1  993      MOV    R0, #LOW(I2CControl) ; Initialize the pointers to be used
15A1 79A6     +1  994      MOV    R1, #LOW(I2CData)
15A3 E2       +1  995      MOVX   A, @R0
+1  996      ; Should first read the EEPROM
15A4 908000   +1  997      MOV    DPTR, #OldeEEPROMdata
15A7 B150     +1  998      CALL    ReadEEPROM          ; Copy 8K bytes
+1  999      ; Now program the new data
15A9 9011F5   +1 1000     MOV    DPTR, #EEPROMheader
15AC 7F1A     +1 1001     MOV    R7, #LOW((EEPROMlength+32)/32)
15AE B170     +1 1002     CALL    WriteEEPROM         ; Write up to 8K bytes in 32byte blocks
+1 1003     ; Check that it worked correctly
15B0 90A000   +1 1004     MOV    DPTR, #NewEEPROMdata
15B3 B150     +1 1005     CALL    ReadEEPROM          ; Check 8K bytes
+1 1006
+1 1007     ; Wait for a Reset
15B5 80FE     +1 1008     Stop:   JMP    Stop
+1 1009
11F5         +1 1010     ORG    USB_ISR - 11        ; Loader will correctly place this
11F5         +1 1011     EEPROMheader:
11F5 B2       +1 1012     DB 0B2H                    ; Cypress-defined ID
11F6 4242     +1 1013     DB 42H, 42H                ; VID = USB Design By Example
11F8 0042     +1 1014     DB 00H, 42H                ; PID = Empty EZ-USB Development Board
11FA 0000     +1 1015     DB 0, 0                    ; Device ID
+1 1016     ; First load module
11FC 03       +1 1017     DB HIGH(ProgramLength)     ; NOTE: Max ProgramLength is 3FFF
11FD 1B       +1 1018     DB LOW(ProgramLength)
11FE 12       +1 1019     DB HIGH(USB_ISR)
11FF 00       +1 1020     DB LOW(USB_ISR)
1200 02       +1 1021     DB 02                        ; This address MUST be USB_ISR (2=LJMP)
+1 1022
0339         +1 1023     EEPROMlength EQU EEPROMend-EEPROMheader
+1 1024
0000         +1 1025     ORG    0
0000 021599   +1 1026     LJMP   ProgEEPROM          ; Overwrite main program start address
+1 1027
+1 1028
+1 1029
```

1030
1031
1032 END