

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

LOC	OBJ	LINE	SOURCE
		1	NAME KeyboardEmulator
		2	; This example converts signals from the real world into "characters
		3	; typed at a keyboard". It also emulates the status LEDs.
		4	;
		5	; Copyright (C) 2001, Intel Corporation
		6	; All rights reserved.
		7	; Permission is hereby granted to merge this program code with other program
		8	; material to create a derivative work. This derivative work may be distributed
		9	; in compiled object form only. Any other publication of this program, in any form,
		10	; without the explicit permission of the copyright holder is prohibited.
		11	;
		12	; Send questions and comments to John.Hyde@intel.com
		13	;
		14	;
		15	; The PC Host believes that it is talking to a PC Keyboard.
		16	; This "trick" is a simple method of interfacing a unique I/O device to
		17	; a PC since no special PC software is required.
		18	; This example inputs from an 8 bit port and generates custom "keystrokes"
		19	; The keystrokes (actually KeyCodes) are stored in a circular buffer and
		20	; sent at the next available IN Report poll time
		21	;
		22	; Derived from BAL Version 3.0
		23	;
		24	; This version works with dScope monSIO0.hex
		25	; (uses Serial Port 0, loads at 1200H)
		26	;
		27	; Changes from Version 2.0
		28	; a) Two misplaced labels corrected
		29	; b) CommandTable moved outside of the constrained page
		30	; c) SETUPDAT buffer copied to direct access memory, simplified coding
		31	; d) Optional Set_Idle now not supported (returns STALL, not ignorred)
		32	; e) R7 used in place of Temp, saved code space
		33	; f) Code reorganized to separate hardware dependant sections
		34	; Old New
		35	; USBINT.A51 Decode.A51
		36	; Vectors.A51 EZInt.A51
		37	; Timer.A51 EZInt.A51
		38	; Main.A51 EZMain.A51
		39	; g) EP0Size made an equate to ease coding of other components
0040		40	EP0Size EQU 64 ; For EZ-USB
		41	; h) Code added for descriptors > EP0Size
		42	;
		43	; Changes from Version 1.0
		44	; a) Register saving removed from Vectors.A51
		45	; Main has no context which needs to be saved
		46	; b) There was a race condition in USB_INT::Set_Report: which
		47	; could cause OLD data to be read. Busy algorithm changed
		48	; c) The USB Version# was incorrectly declared in the Device Descriptor
		49	; It was 0101H and is now 0110H (data from USB IF)
		50	;
		51	;\$include (Declare.A51)
	+1	52	; This module declares the variables and constants used in the examples
	+1	53	; It is common to all of the examples
	+1	54	;
	+1	55	; Declare Special Function Registers used
0088	+1	56	TimerControl DATA 088H
0089	+1	57	TimerMode DATA 089H
008C	+1	58	Timer0High DATA 08CH

```

00A8      +1  59      EI          DATA    0A8H
00E8      +1  60      EIE         DATA    0E8H      ; EZ-USB specific
0091      +1  61      EXIF        DATA    091H      ; EZ-USB specific
00D8      +1  62      EICON       DATA    0D8H      ; EZ-USB specific
0092      +1  63      PageReg     DATA    092H      ; EZ-USB specific, used with MOVX @Ri
0086      +1  64      DPS          DATA    086H      ; EZ-USB specific, used with dual data pointers
0084      +1  65      ODPL        DATA    084H      ; EZ-USB specific, used with dual data pointers
0085      +1  66      ODPH        DATA    085H      ; EZ-USB specific, used with dual data pointers
          +1  67      ;
          +1  68      ; "External" memory locations used, EZ-USB specific
          +1  69      ; Note that most of these variables are in Page 7FH
7FE8      +1  70      SETUPDAT   EQU      07FE8H
7FD4      +1  71      SUDPTR     EQU      07FD4H
7FB4      +1  72      EP0Control EQU      07FB4H
7F00      +1  73      EP0InBuffer EQU     07F00H
7EC0      +1  74      EP0OutBuffer EQU     07EC0H      ; Not in Page 7FH
7E80      +1  75      EP1InBuffer EQU     07E80H      ; Not in Page 7FH
7FB6      +1  76      EP1Control EQU     07FB6H
7FB5      +1  77      IN0ByteCount EQU    07FB5H
7FC5      +1  78      Out0ByteCount EQU    07FC5H
7FB7      +1  79      IN1ByteCount EQU    07FB7H
7FB6      +1  80      IN1CS      EQU     07FB6H
7FAC      +1  81      IN07IEN    EQU     07FACH
7FA9      +1  82      IN07IRQ    EQU     07FA9H
7FAD      +1  83      OUT07IEN    EQU     07FADH
7FAA      +1  84      OUT07IRQ    EQU     07FAAH
7FAE      +1  85      USBIEN     EQU     07FAEH
7FAB      +1  86      USBIRQ     EQU     07FABH
7FD6      +1  87      USBControl EQU     07FD6H
7FA6      +1  88      I2CData     EQU     07FA6H
7FA5      +1  89      I2CControl EQU     07FA5H
7F93      +1  90      PortA_Config EQU     07F93H
7F94      +1  91      PortB_Config EQU     07F94H
7F95      +1  92      PortC_Config EQU     07F95H
7F96      +1  93      PortA_OUT    EQU     07F96H
7F97      +1  94      PortB_OUT    EQU     07F97H
7F98      +1  95      PortC_OUT    EQU     07F98H
7F99      +1  96      PortA_PINS   EQU     07F99H
7F9A      +1  97      PortB_PINS   EQU     07F9AH
7F9B      +1  98      PortC_PINS   EQU     07F9BH
7F9C      +1  99      PortA_OE     EQU     07F9CH
7F9D      +1 100      PortB_OE     EQU     07F9DH
7F9E      +1 101      PortC_OE     EQU     07F9EH
          +1 102      ;
          +1 103      ; Byte Variables
          +1 104
----      +1 105      DSEG       AT 20H
0020      +1 106      FLAGS:      DS         1      ; This register is bit-addressable
          +1 107      ; Bit Variables
          0000      +1 108      Configured   EQU     FLAGS.0 ; Is this device configured
          0001      +1 109      STALL        EQU     FLAGS.1 ; Need to STALL endpoint 0
          0002      +1 110      SendData     EQU     FLAGS.2 ; Need to send data to PC Host
          0003      +1 111      IsDescriptor EQU     FLAGS.3 ; Enable a shortcut reply
          0004      +1 112      SetAddress   EQU     FLAGS.4 ; Set the SIE address
          +1 113      ;
0021      +1 114      MonitorSpace: DS      1FH      ; Used by Dscope
          +1 115      ;Expired_Time: DS      1      ; A downcounter for timed Reports
0040      +1 116      ReplyCount:   DS      1      ; Byte count for following buffer
0041      +1 117      ReplyBuffer:  DS      8      ; Buffer for immediate reply
0049      +1 118      CurrentConfiguration:
0049      +1 119      DS          1      ; Some examples support > 1 configurations
004A      +1 120      SaveDPH:      DS      1      ; Needed to save Descriptor Pointer ..
004B      +1 121      SaveDPL:      DS      1      ; .. for descriptors > EP0Size
004C      +1 122      SaveLength:   DS      1      ; Number of bytes still to send
004D      +1 123      SetupData:    DS      1      ; Buffer in direct access memory
004D      +1 124      RequestType:  DS      1
    
```

```

004E      +1 125      Request:      DS      1
004F      +1 126      wValueLow:     DS      1
0050      +1 127      wValueHigh:    DS      1
0051      +1 128      wIndexLow:     DS      1
0052      +1 129      wIndexHigh:    DS      1
0053      +1 130      wLengthLow:    DS      1
0054      +1 131      wLengthHigh:   DS      1
          +1 132      ;
0055      +1 133      Old_Buttons:   DS      1      ; Stores current button position
0056      +1 134      LEDvalue:      DS      1      ; Stores current LED value
0057      +1 135      Msec_Counter: DS      1      ; Counts up to 4 msec
          +1 136      ;
          +1 137      ; Declare the circular KeyCode buffer
0005      +1 138      EP1INReady  EQU      FLAGS.5
0006      +1 139      BufferOverrun EQU      FLAGS.6
0058      +1 140      KCBufferHead: DS      1
0059      +1 141      KCBufferTail: DS      1
005A      +1 142      KCBuffer:      DS      1      ; EZ-USB has > 7FH bytes. Use a 60 byte buffer
0096      +1 143      KCBufferEnd  EQU      $+59    ; Allow for my big "message"!
          +1 144      ;
          +1 145      ;
          +1 146      ;
          +1 147      ;$include (EZInt.A51)
          +1 148      ; This module contains all the EZUSB-specific hardware code
          +1 149      ; This module also contains all of the interrupt vector declarations and
          +1 150      ; the first level interrupt servicing (register save, call subroutine,
          +1 151      ; clear interrupt source, restore registers, return)
          +1 152      ; Suspend and Resume are handled totally in this module
          +1 153      ;
          +1 154      ; A Reset sends us to Program space location 0
----      +1 155      CSEG AT 0      ; Code space
          +1 156      USING 0      ; Reset forces Register Bank 0
0000 021300 +1 157      LJMP      Reset
          +1 158      ;
          +1 159      ; The interrupt vector table is also located here
          +1 160      ; EZ-USB has two levels of USB interrupts:
          +1 161      ; 1-the main level is described in this table (at ORG 43H)
          +1 162      ; 2-there are 21 sources of USB interrupts and these are described in USB_ISR
          +1 163      ; This means that two levels of acknowledgement and clearing will be required
          +1 164      ;      LJMP      INT0_ISR      ; Features not used are commented out
          +1 165      ;      ORG      0BH
          +1 166      ;      LJMP      Timer0_ISR
          +1 167      ;      ORG      13H
          +1 168      ;      LJMP      INT1_ISR
          +1 169      ;      ORG      1BH
          +1 170      ;      LJMP      Timer1_ISR
          +1 171      ;      ORG      23H
          +1 172      ;      LJMP      UART0_ISR
          +1 173      ;      ORG      2BH
          +1 174      ;      LJMP      Timer2_ISR
          +1 175      ;      ORG      33H
          +1 176      ;      LJMP      WakeUp_ISR
          +1 177      ;      ORG      3BH
          +1 178      ;      LJMP      UART1_ISR
0043      +1 179      ORG      43H
0043 021200 +1 180      LJMP      USB_ISR      ; Auto Vector will replace byte 45H
          +1 181      ;      ORG      4BH
          +1 182      ;      LJMP      I2C_ISR
          +1 183      ;      ORG      53H
          +1 184      ;      LJMP      INT4_ISR
          +1 185      ;      ORG      5BH
          +1 186      ;      LJMP      INT5_ISR
          +1 187      ;      ORG      63H
          +1 188      ;      LJMP      INT6_ISR
          +1 189      ;
1200      +1 190      ORG      1200H      ; Load above monSIO0.hex

```

```

1200 02127C    +1  191    USB_ISR:LJMP    SUDAV_ISR
1203 00        +1  192            DB    0                ; Pad entries to 4 bytes
1204 021266    +1  193            LJMP    SOF_ISR
1207 00        +1  194            DB    0
1208 021223    +1  195            LJMP    SUTOK_ISR
120B 00        +1  196            DB    0
120C 021230    +1  197            LJMP    Suspend_ISR
120F 00        +1  198            DB    0
1210 02122B    +1  199            LJMP    USBReset_ISR
1213 00        +1  200            DB    0
1214 021223    +1  201            LJMP    Reserved
1217 00        +1  202            DB    0
1218 021255    +1  203            LJMP    EP0In_ISR
121B 00        +1  204            DB    0                ; Comment out features not used
121C 021223    +1  205            LJMP    EP0Out_ISR
121F 00        +1  206            DB    0
1220 02123F    +1  207            LJMP    EP1In_ISR
+1  208            ;    DB    0
+1  209            ;    LJMP    EP1Out_ISR
+1  210            ;    DB    0
+1  211            ;    LJMP    EP2In_ISR
+1  212            ;    DB    0
+1  213            ;    LJMP    EP2Out_ISR
+1  214            ;    DB    0
+1  215            ;    LJMP    EP3In_ISR
+1  216            ;    DB    0
+1  217            ;    LJMP    EP3Out_ISR
+1  218            ;    DB    0
+1  219            ;    LJMP    EP4In_ISR
+1  220            ;    DB    0
+1  221            ;    LJMP    EP4Out_ISR
+1  222            ;    DB    0
+1  223            ;    LJMP    EP5In_ISR
+1  224            ;    DB    0
+1  225            ;    LJMP    EP5Out_ISR
+1  226            ;    DB    0
+1  227            ;    LJMP    EP6In_ISR
+1  228            ;    DB    0
+1  229            ;    LJMP    EP6Out_ISR
+1  230            ;    DB    0
+1  231            ;    LJMP    EP7In_ISR
+1  232            ;    DB    0
+1  233            ;    LJMP    EP7Out_ISR
+1  234            ; End of Interrupt Vector tables
+1  235
+1  236            ; When a feature is used insert the required interrupt processing here
+1  237            ; The example use only used Endpoints 0 and 1 and also SOF for timing
1223          +1  238    Reserved:
1223          +1  239    INT0_ISR:
1223          +1  240    Timer0_ISR:
1223          +1  241    INT1_ISR:
1223          +1  242    Timer1_ISR:
1223          +1  243    UART0_ISR:
1223          +1  244    Timer2_ISR:
1223          +1  245    UART1_ISR:
1223          +1  246    I2C_ISR:
1223          +1  247    INT4_ISR:
1223          +1  248    INT5_ISR:
1223          +1  249    INT6_ISR:
1223          +1  250    SUTOK_ISR:
1223          +1  251    EP0Out_ISR:
1223          +1  252    EP1Out_ISR:
1223          +1  253    EP2In_ISR:
1223          +1  254    EP2Out_ISR:
1223          +1  255    EP3In_ISR:
1223          +1  256    EP3Out_ISR:
    
```

```
1223      +1 257      EP4In_ISR:
1223      +1 258      EP4Out_ISR:
1223      +1 259      EP5In_ISR:
1223      +1 260      EP5Out_ISR:
1223      +1 261      EP6In_ISR:
1223      +1 262      EP6Out_ISR:
1223      +1 263      EP7In_ISR :
1223      +1 264      EP7Out_ISR:
1223      +1 265      Not_Used:                ; Should not get any of these
1223 32    +1 266      RETI
          +1 267
1224      +1 268      ClearINT2:                ; Tell the hardware that we're done
1224 E591  +1 269      MOV          A, EXIF
1226 C2E4  +1 270      CLR          ACC.4                ; Clear the Interrupt 2 bit
1228 F591  +1 271      MOV          EXIF, A
122A 22    +1 272      RET
          +1 273
122B      +1 274      USBReset_ISR:            ; Bus has been Reset, move to DEFAULT state
122B 91D7  +1 275      CALL         Deconfigured
122D 5124  +1 276      CALL         ClearINT2
          +1 277                ; No need to clear source of interrupt
122F 32    +1 278      RETI
          +1 279
1230      +1 280      Suspend_ISR:                ; SIE detected an Idle bus
1230 E587  +1 281      MOV          A, PCON
1232 4401  +1 282      ORL          A, #1
1234 F587  +1 283      MOV          PCON, A                ; Go to sleep!
1236 00    +1 284      NOP
1237 00    +1 285      NOP                ; Wake up here due to a USBResume
1238 00    +1 286      NOP
1239 5124  +1 287      CALL         ClearINT2
123B 32    +1 288      RETI
          +1 289
123C      +1 290      WakeUp_ISR:                ; Not using external WAKEUP in these
examples
          +1 291
123C C2DC  +1 292      CLR          EICON.4                ; So this must be due to a USBResume
123E 32    +1 293      RETI                ; Clear the wakeup interrupt source
          +1 294
123F      +1 295      EP1In_ISR:                ; A INPUT Report has just been read
123F D205  +1 296      SETB         EP1INReady            ; Set a Ready Flag
          +1 297                ; Create another report if there is an entry in the KeyCode buffer
1241 C3    +1 298      CLR          C
1242 E559  +1 299      MOV          A, KCBufferTail
1244 9558  +1 300      SUBB         A, KCBufferHead        ; If Head = Tail then buffer is empty
1246 6002  +1 301      JZ          NoKeyCodeAvailable
1248 7181  +1 302      CALL         CreateInputReport
124A      +1 303      NoKeyCodeAvailable:
124A 9002A9 +1 304      MOV          DPTR, #(200H OR LOW(IN07IRQ))
124D      +1 305      ExitISR:                ; Common exit routine for all ISR's
          +1 306                ; On entry DPL = LOW(Interrupt Register), DPH = Interrupt ID
124D 5124  +1 307      CALL         ClearINT2
124F 747F  +1 308      MOV          A, #7FH                ; The EZ-USB I/O register page
1251 C583  +1 309      XCH          A, DPH
1253 F0    +1 310      MOVX         @DPTR, A                ; Clear the current interrupt
1254 32    +1 311      RETI
          +1 312
1255      +1 313      EP0In_ISR:                ; A prepared packet has been read by PC
host
1255 E54C  +1 314      MOV          A, SaveLength            ; Do I have any more data to send?
1257 6008  +1 315      JZ          NoMoreToSend
1259 854A83 +1 316      MOV          DPH, SaveDPH            ; Retrieve descriptor pointer
125C 854B82 +1 317      MOV          DPL, SaveDPL
125F 51C8  +1 318      CALL         SendNextPieceOfDescriptor
1261      +1 319      NoMoreToSend:
1261 9001A9 +1 320      MOV          DPTR, #(100H OR LOW(IN07IRQ))
1264 80E7  +1 321      JMP          ExitISR
          +1 322
```

```
1266      +1 323      SOF_ISR:                                ; A Start-Of-Frame packet has been received
+1 324      ; This routine services the real time interrupt
+1 325      ; It is also responsible for the "real world" buttons
+1 326      ;
1266      +1 327      ServiceTimerRoutine:
1266 D5570E +1 328              DJNZ      Msec_counter, Done      ; Only need to check every 4msec
1269 755704 +1 329              MOV       Msec_counter, #4      ; Reinitialize
+1 330      ;
+1 331      ; Add an entry to the ScanCode buffer if the buttons have changed state
126C      +1 332      ReadButtons:
126C 907F99 +1 333              MOV       DPTR, #PortA_Pins
126F E0     +1 334              MOVX      A, @DPTR
1270 FE     +1 335              MOV       R6, A                                ; Save buttons in case they have changed
1271 6555   +1 336              XRL      A, Old_Buttons
1273 6002   +1 337              JZ       Done
1275 71AB   +1 338              CALL     AddToKeyCodeBuffer
1277 9002AB +1 339      Done:      MOV       DPTR, #(200H OR LOW(USBIRQ))
127A 80D1   +1 340              JMP       ExitISR
+1 341
127C      +1 342      SUDAV_ISR:                                ; A Setup packet has been received
127C 754C00 +1 343              MOV       SaveLength, #0      ; Clear any pending transactions (if any)
127F 907FE8 +1 344              MOV       DPTR, #SETUPDAT      ; Copy packet to direct access memory
1282 784D   +1 345              MOV       R0, #SetupData
1284 7F08   +1 346              MOV       R7, #8
1286 E0     +1 347      CopySD:  MOVX      A, @DPTR
1287 F6     +1 348              MOV       @R0, A
1288 A3     +1 349              INC       DPTR
1289 08     +1 350              INC       R0
128A DFFA   +1 351              DJNZ      R7, CopySD
128C 9108   +1 352              CALL     ServiceSetupPacket      ; Handle the decode of the Setup packet
+1 353      ; if SetAddress { Update SIE address } // NOP on EZ-USB
+1 354      ; if STALL { Stall the endpoint }
+1 355      ; if SendData {
+1 356      ;     if IsDescriptor { send DPTR->descriptor, A = length }
+1 357      ;     else { send ReplyBuffer }
+1 358      ; }
128E 200126 +1 359              JB       STALL, SendSTALL
1291 300216 +1 360              JNB      SendData, HandShake
1294 200324 +1 361              JB       IsDescriptor, LoadEP0
+1 362
+1 363              ; Send data in ReplyBuffer
1297 907F00 +1 363              MOV       DPTR, #EP0InBuffer
129A 7841   +1 364              MOV       R0, #ReplyBuffer
129C 7F08   +1 365              MOV       R7, #8                                ; Copy the max length buffer
129E E6     +1 366      CopyRB:  MOV       A, @R0
129F F0     +1 367              MOVX      @DPTR, A
12A0 A3     +1 368              INC       DPTR
12A1 08     +1 369              INC       R0
12A2 DFFA   +1 370              DJNZ      R7, CopyRB
12A4 E540   +1 371              MOV       A, ReplyCount      ; Get the real length
12A6      +1 372      SendEP0InBuffer:
12A6 907FB5 +1 373              MOV       DPTR, #In0ByteCount
12A9      +1 374      StartXfer:
12A9 F0     +1 375              MOVX      @DPTR, A                                ; This write initiates the transfer
12AA      +1 376      HandShake:
12AA 7F02   +1 377              MOV       R7, #00000010b      ; Set HSNACK to tell the SIE that we're done
12AC      +1 378      SetEP0Control:
12AC 907FB4 +1 379              MOV       DPTR, #EP0Control
12AF E0     +1 380              MOVX      A, @DPTR
12B0 4F     +1 381              ORL      A, R7
12B1 F0     +1 382              MOVX      @DPTR, A                                ; We're done
12B2 9001AB +1 383              MOV       DPTR, #(100H OR LOW(USBIRQ))
12B5 8096   +1 384              JMP       ExitISR
12B7      +1 385      SendSTALL:
12B7 7F03   +1 386              MOV       R7, #00000011b      ; Set EPOSTALL and HSNACK
12B9 80F1   +1 387              JMP       SetEP0Control
12BB      +1 388      LoadEP0:
+1 388              ; Send the data pointed to by DPTR
```

```

12BB FF      +1 389      MOV      R7, A          ; Save LENGTH
+1 390      ; Need to return the smaller of "Requested Length" and "Actual Length"
+1 391      ; If "Requested Length" > 255 then use "Actual Length"
+1 392      ; There are no descriptors > 255 in this example
12BC E554    +1 393      MOV      A, wLengthHigh
12BE 7007    +1 394      JNZ      UseActual
12C0 C3      +1 395      CLR      C
12C1 9553    +1 396      SUBB     A, wLengthLow
12C3 E553    +1 397      MOV      A, wLengthLow      ; This does not affect Carry
12C5 5001    +1 398      JNC      UsewLengthLow
12C7         +1 399      UseActual:
+1 400      MOV      A, R7
12C8         +1 401      UsewLengthLow:
12C8         +1 402      SendNextPieceOfDescriptor: ; DPTR -> Descriptor to be sent
12C8 FF      +1 403      MOV      R7, A          ; Save LENGTH again
12C9 754C00  +1 404      MOV      SaveLength, #0 ; Default case, overwrite if necessary
+1 405      ; Do I have more than a single packet to send?
12CC C3      +1 406      CLR      C
12CD 9440    +1 407      SUBB     A, #EP0Size
12CF 4015    +1 408      JC       SendPacket
+1 409      ; Need to send multiple packets.
+1 410      ; Calculate and save address of next packet, send next packet now
12D1 F54C    +1 411      MOV      SaveLength, A      ; Send these next time
12D3 7F40    +1 412      MOV      R7, #EP0Size
12D5 C083    +1 413      PUSH     DPH          ; Save current pointer
12D7 C082    +1 414      PUSH     DPL
12D9 EF      +1 415      MOV      A, R7          ; Retrieve length
12DA 914B    +1 416      CALL    BumpDPTR
12DC 85834A  +1 417      MOV      SaveDPH, DPH
12DF 85824B  +1 418      MOV      SaveDPL, DPL
12E2 D082    +1 419      POP     DPL
12E4 D083    +1 420      POP     DPH
12E6         +1 421      SendPacket:
12E6 EF      +1 422      MOV      A, R7          ; Retrieve length
12E7 FE      +1 423      MOV      R6, A          ; Save length in R6 for move
12E8 7800    +1 424      MOV      R0, #LOW(EP0InBuffer) ; PageReg = 7FH = HIGH(EP0InBuffer)
12EA E0      +1 425      CopySTD:MOVX     A, @DPTR
12EB F2      +1 426      MOVX     @R0, A
12EC A3      +1 427      INC     DPTR
12ED 08      +1 428      INC     R0
12EE DEFA    +1 429      DJNZ    R6, CopySTD
12F0 EF      +1 430      MOV      A, R7          ; Retrieve LENGTH
12F1 80B3    +1 431      JMP     SendEP0InBuffer
+1 432
12F3         +1 433      GetOutputReport: ; Wait for this, it's next on USB
12F3 907FC5  +1 434      MOV      DPTR, #Out0ByteCount ; Enable EP0OutBuffer to receive data
12F6 F0      +1 435      MOVX     @DPTR, A      ; Any value will do
12F7 907FB4  +1 436      MOV      DPTR, #EP0Control ; Wait for valid data in EP0OutBuffer
12FA E0      +1 437      Wait40:MOVX     A, @DPTR
12FB 5408    +1 438      ANL     A, #00001000b ; Check OUTBSY
12FD 70FB    +1 439      JNZ     Wait40
12FF 22      +1 440      RET
+1 441
+1 442
+1 443
+1 444
+1 445      ;$include (EZMain.A51)
+1 446      ; This module initializes the microcontroller then executes MAIN forever
+1 447      ; It is hardware dependant
+1 448
1300         +1 449      Reset:
1300 7581DF  +1 450      MOV      SP, #0DFH      ; Initialize the Stack
1303 75927F  +1 451      MOV      PageReg, #7FH ; Allows MOVX Ri to access EZ-USB memory
+1 452
1306 78D6    +1 453      MOV      R0, #Low(USBControl) ; Simulate a disconnect
1308 E2      +1 454      MOVX     A, @R0

```

```
1309 54F3      +1  455      ANL    A, #11110011b      ; Clear DISCON, DISCOE
130B F2       +1  456      MOVX   @R0, A
130C 7151     +1  457      CALL  Wait100msec      ; Give the host time to react
130E E2       +1  458      MOVX   A, @R0          ; Reconnect with this new identity
130F 4406     +1  459      ORL    A, #00000110b   ; Set DISCOE to enable pullup resistor
1311 F2       +1  460      MOVX   @R0, A          ; Set RENUM so that 8051 handles USB
requests
1312 E4       +1  461      CLR    A
1313 F520     +1  462      MOV    FLAGS, A       ; Start in Default state
1315         +1  463      TurnOffLEds:
1315 F556     +1  464      MOV    LEDValue, A
1317 F555     +1  465      MOV    Old_Buttons, A
1319         +1  466      Initialize4msecCounter:
1319 04       +1  467      INC    A               ; = 1
131A F557     +1  468      MOV    Msec_counter, A
131C         +1  469      InitializeKeyCodeBuffer:
131C 75595A   +1  470      MOV    KCBufferTail, #KCBuffer
131F 75585A   +1  471      MOV    KCBufferHead, #KCBuffer
1322 7175     +1  472      CALL  ClearReport
1324         +1  473      InitializeIOSystem:   ; Setup for Simmbus A=input, B=output
+1  474      ; C=External RD#,WR#,TD0,TR0
1324 7893     +1  475      MOV    R0, #LOW(PortA_Config) ; PageReg = 7F = HIGH(PortA_Config)
1326 799C     +1  476      MOV    R1, #LOW(PortA_OE)
1328 E4       +1  477      CLR    A
1329 F2       +1  478      MOVX   @R0, A          ; No alternate functions
132A F3       +1  479      MOVX   @R1, A          ; Enable PortA for Input
132B 08       +1  480      INC    R0             ; Point to PortB_Config
132C 09       +1  481      INC    R1             ; Point to PortB_OE
132D F2       +1  482      MOVX   @R0, A          ; No alternate functions
132E F4       +1  483      CPL    A              ; = 0FFH
132F F3       +1  484      MOVX   @R1, A          ; Enable PortB for Output
1330 08       +1  485      INC    R0             ; Point to PortC_Config
1331 09       +1  486      INC    R1             ; Point to PortC_OE
1332 74C3     +1  487      MOV    A, #11000011b   ;
1334 F2       +1  488      MOVX   @R0, A          ; Alternate functions on [7,6,1,0]
1335 74C2     +1  489      MOV    A, #11000010b   ;
1337 F3       +1  490      MOVX   @R1, A          ; Most alternate functions are outputs
1338         +1  491      InitializeInterruptSystem: ; First initialize the USB level
1338 7401     +1  492      MOV    A, #00000001b
133A 78AC     +1  493      MOV    R0, #LOW(IN07IEN)
133C F2       +1  494      MOVX   @R0, A          ; Enable interrupts from EP0IN
133D 08       +1  495      INC    R0
133E E4       +1  496      CLR    A
133F F2       +1  497      MOVX   @R0, A          ; Disable interrupts from OUT Endpoints 0-7
1340 08       +1  498      INC    R0
1341 7403     +1  499      MOV    A, #00000011b
1343 F2       +1  500      MOVX   @R0, A          ; Enable (Resume, Suspend,) SOF and SUDAV
INTs
1344 08       +1  501      INC    R0
1345 7401     +1  502      MOV    A, #00000001b
1347 F2       +1  503      MOVX   @R0, A          ; Enable Auto Vectoring for USB interrupts
+1  504
1348 75E801   +1  505      MOV    EIE, #00000001b ; Enable INT2 = USB Interrupt (only)
134B 75A890   +1  506      MOV    EI, #10010000b  ; Enable interrupt subsystem (and Ser0 for
dScope)
+1  507
134E         +1  508      MAIN:
134E 00       +1  509      NOP                                ; Not much of a main loop for this example
134F 80FD     +1  510      JMP    MAIN                      ; All actions are initiated by interrupts
+1  511      ; We are a slave, we wait to be told what to do
+1  512
1351         +1  513      Wait100msec:
1351 7F64     +1  514      MOV    R7, #100
1353         +1  515      Wait1msec:                ; A delay loop
1353 758600   +1  516      MOV    DPS, #0             ; Select primary DPTR
1356 90FB50   +1  517      MOV    DPTR, #-1200
1359 A3       +1  518      More:    INC    DPTR          ; 3 cycles
135A E582     +1  519      MOV    A, DPL             ; + 2
135C 4583     +1  520      ORL    A, DPH             ; + 2
```

```
135E 70F9      +1 521          JNZ     More                ; + 3 = 10 cycles x 1200 = 1msec
1360 DFF1      +1 522          DJNZ   R7, Wait1msec
1362 22        +1 523          RET
1363          +1 524
1363          +1 525          ProcessOutputReport:      ; A Report has just been received
1363          +1 526          ; The report is only one byte long in this example
1363          +1 527          ; It contains a new value for the LEDs
1363 907EC0    +1 528          MOV     DPTR, #EP0OutBuffer ; Point to the Report
1366 E0        +1 529          MOVX   A, @DPTR            ; Get the Data
1367 F556      +1 530          MOV     LEDValue, A        ; Update the local variable
1369 907F97    +1 531          MOV     DPTR, #PortB_Out   ;
136C F0        +1 532          MOVX   @DPTR, A           ; Update the "real" LED's
136D 22        +1 533          RET
136D          +1 534
136D          +1 535          ; Manage a circular buffer
136E          +1 536          BumpBufferPtr:
136E 08        +1 537          INC     R0
136F B89602    +1 538          CJNE   R0, #KCBufferEnd, NoWrap
1372 785A      +1 539          MOV     R0, #KCBuffer
1374 22        +1 540          NoWrap: RET
1374          +1 541
1375          +1 542          ClearReport:
1375 907E88    +1 543          MOV     DPTR, #EP1InBuffer+8 ; Point to the end of the buffer
1378 E4        +1 544          CLR     A                 ; Clear it backwards
1379 7F08      +1 545          MOV     R7, #8            ; Length of report
137B F0        +1 546          ClearR: MOVX   @DPTR, A        ; Clear Report
137C 1582      +1 547          DEC     DPL               ; This is safe
137E DFFB      +1 548          DJNZ   R7, ClearR
1380 22        +1 549          RET
1380          +1 550
1381          +1 551          CreateInputReport:
1381          +1 552          ; Create a standard keyboard input report
1381          +1 553          ; 8 bytes = Modifier, 0, Keycode(6)
1381          +1 554          ;
1381          +1 555          ; Modifier(Left Shift) is set dependant upon LEDValue(ShiftLock)
1381          +1 556          ; Keycode(0) is set to the value in the ScanCode buffer
1381 7175      +1 557          CALL   ClearReport       ; Returns with DPTR -> IN1Buffer
1383 E556      +1 558          MOV     A, LEDValue
1385 5402      +1 559          ANL    A, #00000010b     ; Select CapsLock bit
1387 F0        +1 560          MOVX   @DPTR, A         ; Set Modifier byte (LeftShift in bit 1!)
1388 A3        +1 561          INC     DPTR
1389 A3        +1 562          INC     DPTR            ; Point to KeyCode(0)
138A A859      +1 563          MOV     R0, KCBufferTail
138C E6        +1 564          MOV     A, @R0          ; Retrieve KeyCode
138D F0        +1 565          MOVX   @DPTR, A         ; Set KeyCode(0)
138E 716E      +1 566          CALL   BumpBufferPtr
1390 8859      +1 567          MOV     KCBufferTail, R0
1392          +1 568          SendReport:
1392 907FB7    +1 569          MOV     DPTR, #IN1ByteCount
1395 7408      +1 570          MOV     A, #8
1397 F0        +1 571          MOVX   @DPTR, A         ; Endpoint 1 now 'armed', next IN will get
data
1398 C205      +1 572          CLR     EP1InReady
139A 22        +1 573          RET
139A          +1 574
139A          +1 575          ; Add entry to KeyCode buffer
139B          +1 576          AddKeyOff:
139B E4        +1 577          CLR     A
139C          +1 578          AddEntry:
139C A858      +1 579          MOV     R0, KCBufferHead
139E F6        +1 580          MOV     @R0, A
139F 716E      +1 581          CALL   BumpBufferPtr    ; Check that we're not over-writing the
Tail
13A1 E8        +1 582          MOV     A, R0
13A2 B55903   +1 583          CJNE   A, KCBufferTail, WriteOK; If Head = Tail now, then we have an
overflow
13A5 D206      +1 584          SETB   BufferOverrun    ; Don't update Head pointer
13A7 22        +1 585          RET
13A8 8858      +1 586          WriteOK:MOV    KCBufferHead, R0 ; OK to update Head pointer
```

```

13AA 22      +1  587          RET
              +1  588
13AB         +1  589      AddToKeyCodeBuffer:
              +1  590      ; This example looks at the 8 bits of Port A: and generates a different
              +1  591      ; keycode on LOtoHI and HIToLO transitions.
              +1  592      ; The KeyBoard LED value is checked (must be set via another keyboard)
              +1  593      ; CapsLock generates SHIFT in the Modifier byte
              +1  594      ; ScrollLock generates a string for each transition
              +1  595      ;
13AB FD      +1  596          MOV     R5, A          ; Save the change pattern
13AC EE      +1  597          MOV     A, R6          ; Get New_Buttons
13AD F555    +1  598          MOV     Old_Buttons, A      ; Update Old_Buttons
              +1  599      ; First check if ScrollLock is Set
13AF E556    +1  600          MOV     A, LEDvalue
13B1 5404    +1  601          ANL     A, #00000100b
13B3 6010    +1  602          JZ      NoScrollLock
13B5         +1  603      ScrollLock:                ; Send a message
13B5 7F00    +1  604          MOV     R7, #0
13B7 EF      +1  605      SLLoop: MOV     A, R7
13B8 9013EF  +1  606          MOV     DPTR, #Message
13BB 93      +1  607          MOVC   A, @A + DPTR
13BC 601D    +1  608          JZ      AddDone
13BE 719C    +1  609          CALL   AddEntry
13C0 719B    +1  610          CALL   AddKeyOff
13C2 0F      +1  611          INC     R7
13C3 80F2    +1  612          JMP     SLLoop
              +1  613      ; Determine which button was set/reset
13C5         +1  614      NoScrollLock:
13C5 ED      +1  615          MOV     A, R5          ; Get change pattern
13C6 7FFF    +1  616          MOV     R7, #-1       ; Set up a bit counter
13C8 0F      +1  617      Next: INC     R7          ; Point to bit being tested
13C9 13      +1  618          RRC     A          ; Test change status
13CA 50FC    +1  619          JNC     Next
              +1  620      ; Found the changed bit, R7 has its index value
13CC EE      +1  621          MOV     A, R6          ; Get New_Buttons
13CD 5D      +1  622          ANL     A, R5          ; Check against change pattern
13CE 6002    +1  623          JZ      HIToLO       ; 0 = New Button is LO
13D0 7408    +1  624          MOV     A, #8          ; Index into Response table
13D2         +1  625      HIToLO:
13D2 2F      +1  626          ADD     A, R7          ; Add index
              +1  627      ; Return the keycode from the lookup table
13D3 9013DF  +1  628          MOV     DPTR, #Response
13D6 93      +1  629          MOVC   A, @A + DPTR
13D7 719C    +1  630          CALL   AddEntry
13D9 719B    +1  631          CALL   AddKeyOff
              +1  632      ; If EPl IN Buffer is available, send the first report
13DB 2005A3  +1  633      AddDone:JB     EPlINReady, CreateInputReport
13DE 22      +1  634          RET
              +1  635
              +1  636      ; Declare the keycode lookup table
13DF         +1  637      Response:
              +1  638      ; Low to High button transition
13DF 17081617 +1  639          DB     17H,08H,16H,17H,0CH,11H,0AH,2CH ; 'testing '
13E3 0C110A2C +1  640      ; High to Low button transition
13E7 1E1F2021 +1  641          DB     1EH,1FH,20H,21H,22H,23H,24H,25H ; '12345678'
13EB 22232425 +1  642      Message:
13EF         +1  643      ; Any transition generates the same message
13EF 28181605 +1  644          DB     28H,18H,16H,05H,2CH,07H,08H,16H ; '|usb des'
13F3 2C070816 +1  645          DB     0CH,0AH,11H,2CH,05H,1CH,2CH,08H ; 'ign by e'
13F7 0C0A112C +1  646          DB     1BH,04H,10H,13H,0FH,08H,28H,28H ; 'xample||'
13FB 051C2C08 +1  647          DB     0
13FF 1B041013 +1  647          DB     0
1403 0F082828 +1  647          DB     0
1407 00      +1  647          DB     0          ; EOM marker

```

```

+1 648
+1 649
+1 650
+1 651
652
653 ;$include (Decode.A51)
+1 654 ; This module is common to all of the examples.
+1 655 ; It decodes the USB Setup Packets and generates appropriate responses.
+1 656 ; Interpretation of Reports is handled by MAIN
+1 657 ;
----
+1 658 CSEG
1408 +1 659 ServiceSetupPacket:
1408 E54D +1 660 MOV A, RequestType
140A A2E7 +1 661 MOV C, ACC.7 ; Bit 7 = 1 means IO device needs to send
data to P

C Host
140C 9202 +1 662 MOV SendData, C
140E 545C +1 663 ANL A, #01011100b ; IF RequestType[6.4.3.2] = 1 THEN goto
BadRequest
1410 7035 +1 664 JNZ BadRequest
1412 E54D +1 665 MOV A, RequestType ; IF RequestType[1&0] = 1 THEN goto
BadRequest
1414 A2E0 +1 666 MOV C, ACC.0
1416 82E1 +1 667 ANL C, ACC.1
1418 402D +1 668 JC BadRequest
141A 30E502 +1 669 JNB ACC.5, NotB5 ; IF RequestType[5] = 1 THEN
RequestType[1,0] = [1,
1]
141D 7403 +1 670 MOV A, #00000011b
141F 5403 +1 671 NotB5: ANL A, #00000011b ; Set CommandIndex[5,4] = RequestType[1,0]
1421 C4 +1 672 SWAP A
1422 FF +1 673 MOV R7, A ; Save HI nibble of CommandIndex
+1 674 ; Set CommandIndex[3,0] = Request[3,0]
1423 E54E +1 675 MOV A, Request
1425 54F0 +1 676 ANL A, #11110000b ; Check if Request > 15
1427 701E +1 677 JNZ BadRequest
1429 E54E +1 678 MOV A, Request
142B 540F +1 679 ANL A, #00001111b ; Only 13 are defined today, handle in
table
142D 4F +1 680 ORL A, R7
+1 681 ; goto CommandTable(CommandIndex)
142E +1 682 CorrectSubroutine: ; Jump to the subroutine that DPTR is
pointing to
142E 754001 +1 683 MOV ReplyCount, #1 ; Set up a default reply
1431 754100 +1 684 MOV ReplyBuffer, #0
1434 754200 +1 685 MOV ReplyBuffer+1, #0
1437 C204 +1 686 CLR SetAddress ; Clear all flags
1439 C201 +1 687 CLR STALL
143B C203 +1 688 CLR IsDescriptor
143D 901454 +1 689 MOV DPTR, #CommandTable
1440 914B +1 690 CALL BumpDPTR ; Point to entry
1442 E0 +1 691 MOVX A, @DPTR ; Get the offset
1443 901494 +1 692 MOV DPTR, #Subroutines
1446 73 +1 693 JMP @A+DPTR ; Go to the correct Subroutine
+1 694
1447 +1 695 BadRequest: ; Decoded a Bad Request, STALL the Endpoint
1447 D201 +1 696 SETB STALL
1449 22 +1 697 RET
+1 698 ; Support routines
144A +1 699 NextDPTR: ; Returns (DPTR + byte DPTR is pointing to)
144A E0 +1 700 MOVX A, @DPTR
144B +1 701 BumpDPTR: ; Returns (DPTR + ACC)
144B 2582 +1 702 ADD A, DPL ; Assumes Primary DPTR
144D F582 +1 703 MOV DPL, A
144F 5002 +1 704 JNC Skip
1451 0583 +1 705 INC DPH ; Need 16 bit arithmetic here
1453 22 +1 706 Skip: RET
+1 707
+1 708 ; Since the table only contains byte offsets, it is important that all these
routines are
+1 709 ; within one page (256 bytes) of Subroutines:
+1 710 ; V3.0 - CommandTable moved outside of this one page limited space
1454 +1 711 CommandTable:

```

```

        +1 712      ; First 16 commands are for the Device
1454 25      +1 713      DB LOW(Device_Get_Status - Subroutines)
1455 00      +1 714      DB LOW(Device_Clear_Feature - Subroutines)
1456 00      +1 715      DB LOW(Invalid - Subroutines)
1457 00      +1 716      DB LOW(Device_Set_Feature - Subroutines)
1458 00      +1 717      DB LOW(Invalid - Subroutines)
1459 03      +1 718      DB LOW(Set_Address - Subroutines)
145A 52      +1 719      DB LOW(Get_Descriptor - Subroutines)
145B 00      +1 720      DB LOW(Set_Descriptor - Subroutines)
145C 21      +1 721      DB LOW(Get_Configuration - Subroutines)
145D 2D      +1 722      DB LOW(Set_Configuration - Subroutines)
145E 00      +1 723      DB LOW(Invalid - Subroutines)
145F 00      +1 724      DB LOW(Invalid - Subroutines)
1460 00      +1 725      DB LOW(Invalid - Subroutines)
1461 00      +1 726      DB LOW(Invalid - Subroutines)
1462 00      +1 727      DB LOW(Invalid - Subroutines)
1463 00      +1 728      DB LOW(Invalid - Subroutines)
        +1 729      ; Next 16 commands are for the Interface
1464 29      +1 730      DB LOW(Interface_Get_Status - Subroutines)
1465 00      +1 731      DB LOW(Interface_Clear_Feature - Subroutines)
1466 00      +1 732      DB LOW(Invalid - Subroutines)
1467 00      +1 733      DB LOW(Interface_Set_Feature - Subroutines)
1468 00      +1 734      DB LOW(Invalid - Subroutines)
1469 00      +1 735      DB LOW(Invalid - Subroutines)
146A 79      +1 736      DB LOW(Get_Class_Descriptor - Subroutines)
146B 00      +1 737      DB LOW(Set_Class_Descriptor - Subroutines)
146C 00      +1 738      DB LOW(Invalid - Subroutines)
146D 00      +1 739      DB LOW(Invalid - Subroutines)
146E 00      +1 740      DB LOW(Get_Interface - Subroutines)
146F 00      +1 741      DB LOW(Set_Interface - Subroutines)
1470 00      +1 742      DB LOW(Invalid - Subroutines)
1471 00      +1 743      DB LOW(Invalid - Subroutines)
1472 00      +1 744      DB LOW(Invalid - Subroutines)
1473 00      +1 745      DB LOW(Invalid - Subroutines)
        +1 746      ; Next 16 commands are for the Endpoint
1474 29      +1 747      DB LOW(Endpoint_Get_Status - Subroutines)
1475 00      +1 748      DB LOW(Endpoint_Clear_Feature - Subroutines)
1476 00      +1 749      DB LOW(Invalid - Subroutines)
1477 00      +1 750      DB LOW(Endpoint_Set_Feature - Subroutines)
1478 00      +1 751      DB LOW(Invalid - Subroutines)
1479 00      +1 752      DB LOW(Invalid - Subroutines)
147A 00      +1 753      DB LOW(Invalid - Subroutines)
147B 00      +1 754      DB LOW(Invalid - Subroutines)
147C 00      +1 755      DB LOW(Invalid - Subroutines)
147D 00      +1 756      DB LOW(Invalid - Subroutines)
147E 00      +1 757      DB LOW(Invalid - Subroutines)
147F 00      +1 758      DB LOW(Invalid - Subroutines)
1480 00      +1 759      DB LOW(Endpoint_Sync_Frame - Subroutines)
1481 00      +1 760      DB LOW(Invalid - Subroutines)
1482 00      +1 761      DB LOW(Invalid - Subroutines)
1483 00      +1 762      DB LOW(Invalid - Subroutines)
        +1 763      ; Next 16 commands are Class Requests
1484 00      +1 764      DB LOW(Invalid - Subroutines)
1485 0D      +1 765      DB LOW(Get_Report - Subroutines)
1486 00      +1 766      DB LOW(Get_Idle - Subroutines)
1487 00      +1 767      DB LOW(Get_Protocol - Subroutines)
1488 00      +1 768      DB LOW(Invalid - Subroutines)
1489 00      +1 769      DB LOW(Invalid - Subroutines)
148A 00      +1 770      DB LOW(Invalid - Subroutines)
148B 00      +1 771      DB LOW(Invalid - Subroutines)
148C 00      +1 772      DB LOW(Invalid - Subroutines)
148D 06      +1 773      DB LOW(Set_Report - Subroutines)
148E 00      +1 774      DB LOW(Set_Idle - Subroutines)
148F 00      +1 775      DB LOW(Set_Protocol - Subroutines)
1490 00      +1 776      DB LOW(Invalid - Subroutines)
1491 00      +1 777      DB LOW(Invalid - Subroutines)
    
```

```

1492 00      +1  778          DB LOW(Invalid - Subroutines)
1493 00      +1  779          DB LOW(Invalid - Subroutines)
              +1  780
1494         +1  781      Subroutines:
              +1  782      ;
              +1  783      ; Many requests are INVALID for this example
1494         +1  784      Get_Protocol:          ; We are not a Boot device
1494         +1  785      Set_Protocol:          ; We are not a Boot device
1494         +1  786      Set_Descriptor:        ; Our Descriptors are static
1494         +1  787      Set_Class_Descriptor:  ; Our Descriptors are static
1494         +1  788      Set_Interface:         ; We only have one Interface
1494         +1  789      Get_Interface:         ; We do not have an Alternate setting
1494         +1  790      Set_Idle:              ; V3.0 Optional command, not supported
1494         +1  791      Get_Idle:              ; V3.0 Optional command, not supported
1494         +1  792      Device_Set_Feature:    ; We have no features that can be set or cleared
1494         +1  793      Interface_Set_Feature: ; We have no features that can be set or cleared
1494         +1  794      Endpoint_Set_Feature:  ; We have no features that can be set or cleared
1494         +1  795      Endpoint_Clear_Feature: ; V3.0 We have no features that can be set or
cleared
1494         +1  796      Device_Clear_Feature:  ; We have no features that can be set or cleared
1494         +1  797      Interface_Clear_Feature: ; We have no features that can be set or cleared
1494         +1  798      Endpoint_Sync_Frame:    ; We are not an Isonchronous device
              +1  799
1494         +1  800      Invalid:                ; Invalid Request made, STALL the Endpoint
1494 D201     +1  801          SETB      STALL
1496 22      +1  802      Reply:  RET
              +1  803
1497         +1  804      Set_Address:           ; Set the address that the SIE will respond to
1497 D204     +1  805          SETB      SetAddress
1499 22      +1  806          RET
              +1  807
149A        +1  808      Set_Report:             ; Host wants to sent us a Report.
              +1  809      ; The ONLY case in this example where host sends data to us
149A 3000F7  +1  810          JNB      Configured, Invalid ; Need to be Configured to do this command
149D 51F3    +1  811          CALL    GetOutputReport ; Handled in EZUSB.A51
149F 6163    +1  812          JMP      ProcessOutputReport ; RETurn via this subroutine
14A1        +1  813      Get_Report:           ; Host wants a Report
14A1 3000F0  +1  814          JNB      Configured, Invalid ; Need to be Configured to do this command
              +1  815      ; Copy to latest report into the ReplyBuffer
14A4 907E80  +1  816          MOV      DPTR, #EPlInBuffer
14A7 7841    +1  817          MOV      R0, #ReplyBuffer
14A9 7F08    +1  818          MOV      R7, #8
14AB E0      +1  819      GRLoop: MOVX     A, @DPTR
14AC F6      +1  820          MOV      @R0, A
14AD A3      +1  821          INC     DPTR
14AE 08      +1  822          INC     R0
14AF DFFA    +1  823          DJNZ   R7, GRLoop
14B1 754008  +1  824          MOV      ReplyCount, #8
14B4 22      +1  825          RET
14B5        +1  826      Get_Configuration:         ; Respond with CurrentConfiguration
14B5 854941  +1  827          MOV      ReplyBuffer, CurrentConfiguration
14B8 22      +1  828          RET
14B9        +1  829      Device_Get_Status:       ; Only two bits of Device Status are
defined
14B9 754101  +1  830          MOV      ReplyBuffer, #1 ; Bit 1=Remote Wakeup(=0), Bit 0=Self
Powered(=1)
14BC 22      +1  831          RET
14BD        +1  832      Interface_Get_Status:     ; Interface Status is currently defined as
0
14BD        +1  833      Endpoint_Get_Status:
14BD 754002  +1  834          MOV      ReplyCount, #2 ; Need a two byte 0 response
14C0 22      +1  835          RET
14C1        +1  836      Set_Configuration:         ; Valid values are 0 and 1
14C1 E54F    +1  837          MOV      A, wValueLow
14C3 6012    +1  838          JZ      Deconfigured
14C5 14      +1  839          DEC     A
14C6 70CC    +1  840          JNZ   Invalid
14C8 D200    +1  841          SETB   Configured
14CA 754901  +1  842          MOV      CurrentConfiguration, #1
              +1  843      ; It is now OK to generate Input Reports

```

```

+1 844 ; Enable generation of Input Reports
14CD 907FAC +1 845 MOV DPTR, #IN07IEN
14D0 E0 +1 846 MOVX A, @DPTR
14D1 4402 +1 847 ORL A, #00000010b ; Enable EPIIN interrupts
14D3 F0 +1 848 MOVX @DPTR, A
14D4 D205 +1 849 SETB EPIInReady
14D6 22 +1 850 RET
14D7 +1 851 Deconfigured:
14D7 C200 +1 852 CLR Configured
14D9 754900 +1 853 MOV CurrentConfiguration, #0
+1 854 ; Disable generation of Input Reports
14DC 907FAC +1 855 MOV DPTR, #IN07IEN
14DF E0 +1 856 MOVX A, @DPTR
14E0 54FD +1 857 ANL A, #11111101b ; Disable EPIIN interrupts
14E2 F0 +1 858 MOVX @DPTR, A
14E3 C205 +1 859 CLR EPIInReady
14E5 22 +1 860 RET
14E6 +1 861 Get_Descriptor: ; Host wants to know who/what we are
14E6 D203 +1 862 SETB IsDescriptor
14E8 E550 +1 863 MOV A, wValueHigh
14EA 14 +1 864 DEC A ; Valid Values are 1, 2 and 3
14EB 901526 +1 865 MOV DPTR, #DeviceDescriptor
14EE 6031 +1 866 JZ ReturnLength
14F0 14 +1 867 DEC A
14F1 901538 +1 868 MOV DPTR, #ConfigurationDescriptor
14F4 7003 +1 869 JNZ TryString
14F6 7422 +1 870 MOV A, #ConfigLength
14F8 22 +1 871 RET
14F9 +1 872 TryString:
14F9 14 +1 873 DEC A
14FA 7098 +1 874 JNZ Invalid
+1 875 ; Request is for a String Descriptor
14FC 901595 +1 876 MOV DPTR, #String0 ; Point to String 0
14FF E54F +1 877 MOV A, wValueLow ; Get String Index
1501 +1 878 NextString:
1501 601E +1 879 JZ ReturnLength
1503 FF +1 880 MOV R7, A ; Save String Index
1504 914A +1 881 CALL NextDPTR
1506 E0 +1 882 MOVX A, @DPTR ; Get the String Length (= 0 means we're at
Backsto
p)
1507 608B +1 883 JZ Invalid ; Asked for a string I don't have
1509 EF +1 884 MOV A, R7
150A 14 +1 885 DEC A
150B 80F4 +1 886 JMP NextString ; Check if we are there yet
150D +1 887 Get_Class_Descriptor: ; Valid values are 21H, 22H, 23H for Class
Request
150D D203 +1 888 SETB IsDescriptor
150F E550 +1 889 MOV A, wValueHigh
1511 C3 +1 890 CLR C
1512 9421 +1 891 SUBB A, #21H
1514 90154A +1 892 MOV DPTR, #HIDDescriptor
1517 6008 +1 893 JZ ReturnLength
1519 14 +1 894 DEC A
151A 90155A +1 895 MOV DPTR, #ReportDescriptor
151D 6004 +1 896 JZ ReturnRDlength
+1 897 ; DEC A ; This example does not use Physical
Descriptors
+1 898 ; JZ Send_Physical_Descriptor
151F 8194 +1 899 JMP Invalid
+1 900 ;
1521 +1 901 ReturnLength:
1521 E0 +1 902 MOVX A, @DPTR ; Get Descriptor Length (first byte)
1522 22 +1 903 RET
1523 +1 904 ReturnRDlength: ; Report Descriptor is different format
1523 743B +1 905 MOV A, #ReportLength
1525 22 +1 906 RET
+1 907 ; Error check: this MUST be on within a page of Subroutines
0092 +1 908 WithinSamePage EQU $ - Subroutines

```

```

+1 909 ;
+1 910
+1 911
+1 912 ;$include (DTables.A51)
+1 913 ; This module declares the descriptors
+1 914 ;
+1 915 ; Use a standard keyboard descriptor
+1 916 ;
+1 917 ; This example has one Device Descriptor with:
+1 918 ;     One Configuration - single IN port and single OUT port
+1 919 ;     One Interface - there is only one method of accessing the ports
+1 920 ;     One HID Descriptor - to make PC host software simpler
+1 921 ;     One Endpoint Descriptor - for HID Input Reports
+1 922 ;     One Report Descriptor - one byte IN and one byte OUT reports
+1 923 ;     Multiple Sting Descriptors - to aid the user
+1 924 ;
----
+1 925         CSEG
1526 +1 926 DeviceDescriptor:
1526 1201 +1 927         DB     18, 1           ; Length, Type
1528 1001 +1 928         DB     10H, 1          ; USB Rev 1.1 (=0110H, low=10H, High=01H)
152A 000000 +1 929         DB     0, 0, 0          ; Class, Subclass and Protocol
152D 40 +1 930         DB     EPOSize
152E 42420242 +1 931         DB     42H, 42H, 2, 42H, 0, 1; Vendor ID, Product ID and Version
1532 0001
1534 010200 +1 932         DB     1, 2, 0          ; Manufacturer, Product & Serial# Names
1537 01 +1 933         DB     1           ; #Configs
1538 +1 934 ConfigurationDescriptor:
1538 0902 +1 935         DB     9, 2           ; Length, Type
153A 2200 +1 936         DB     LOW(ConfigLength), HIGH(ConfigLength)
153C 010100 +1 937         DB     1, 1, 0          ; #Interfaces, Configuration#, Config. Name
153F 80 +1 938         DB     10000000b      ; Attributes = Bus Powered
1540 FA +1 939         DB     250          ; Max. Power is 250x2 = 500mA
1541 +1 940 InterfaceDescriptor:
1541 0904 +1 941         DB     9, 4           ; Length, Type
1543 000001 +1 942         DB     0, 0, 1          ; No alternate setting, HID uses EP1
1546 03 +1 943         DB     3           ; Class = Human Interface Device
1547 0000 +1 944         DB     0, 0          ; Subclass and Protocol
1549 00 +1 945         DB     0           ; Interface Name
154A +1 946 HIDDescriptor:
154A 0921 +1 947         DB     9, 21H          ; Length, Type
154C 0001 +1 948         DB     0, 1          ; HID Class Specification compliance
154E 00 +1 949         DB     0           ; Country localization (=none)
154F 01 +1 950         DB     1           ; Number of descriptors to follow
1550 22 +1 951         DB     22H          ; And it's a Report descriptor
1551 3B00 +1 952         DB     LOW(ReportLength), HIGH(ReportLength)
1553 +1 953 EndpointDescriptor:
1553 0705 +1 954         DB     7, 5           ; Length, Type
1555 81 +1 955         DB     10000001b      ; Address = IN 1
1556 03 +1 956         DB     00000011b     ; Interrupt
1557 4000 +1 957         DB     EPOSize, 0      ; Maximum packet size
1559 14 +1 958         DB     20           ; Poll every 20msec seconds
    0022 +1 959         ConfigLength EQU $ - ConfigurationDescriptor
+1 960
155A +1 961 ReportDescriptor:
+1 962 ; Standard USB Keyboard
155A 0501 +1 963         DB     5, 1           ; Usage_Page (Generic Desktop)
155C 0906 +1 964         DB     9, 6           ; Usage (Keyboard)
155E A101 +1 965         DB     0A1H, 1        ; Collection (Application)
1560 0507 +1 966         DB     5, 7           ; Usage page (Key Codes)
1562 19E0 +1 967         DB     19H, 224      ; Usage_Minimum (224)
1564 29E7 +1 968         DB     29H, 231      ; Usage_Maximum (231)
1566 1500 +1 969         DB     15H, 0          ; Logical_Minimum (0)
1568 2501 +1 970         DB     25H, 1        ; Logical_Maximum (1)
156A 7501 +1 971         DB     75H, 1        ; Report_Size (1)
156C 9508 +1 972         DB     95H, 8          ; Report_Count (8)
156E 8102 +1 973         DB     81H, 2          ; Input (Data,Var,Abs) = Modifier Byte

```

```

1570 8101      +1  974          DB      81H, 1          ; Input (Constant) = Reserved Byte
1572 1900      +1  975          DB      19H, 0          ; Usage_Minimum (0)
1574 2965      +1  976          DB      29H, 101         ; Usage_Maximum (101)
1576 1500      +1  977          DB      15H, 0          ; Logical_Minimum (0)
1578 2565      +1  978          DB      25H, 101         ; Logical_Maximum (101)
157A 7508      +1  979          DB      75H, 8          ; Report_Size (8)
157C 9506      +1  980          DB      95H, 6          ; Report_Count (6)
157E 8100      +1  981          DB      81H, 0          ; Input (Data,Array) = Keycode Bytes(6)
1580 0508      +1  982          DB      5, 8           ; Usage_Page (LEDs)
1582 1901      +1  983          DB      19H, 1          ; Usage_Minimum (1)
1584 2905      +1  984          DB      29H, 5          ; Usage_Maximum (5)
1586 1500      +1  985          DB      15H, 0          ; Logical_Minimum (0)
1588 2501      +1  986          DB      25H, 1          ; Logical_Maximum (1)
158A 7501      +1  987          DB      75H, 1          ; Report_Size (1)
158C 9505      +1  988          DB      95H, 5          ; Report_Count (5)
158E 9102      +1  989          DB      91H, 2          ; Output (Data,Var,Abs) = LEDs (5 bits)
1590 9503      +1  990          DB      95H, 3          ; Report_Count (3)
1592 9101      +1  991          DB      91H, 1          ; Output (Constant) = Pad (3 bits)
1594 C0        +1  992          DB      0C0H         ; End_Collection
   003B        +1  993          ReportLength EQU $-ReportDescriptor
   +1  994
1595          +1  995          String0:          ; Declare the UNICODE strings
1595 04030904  +1  996          DB      4, 3, 9, 4      ; Only English language strings supported
1599          +1  997          String1:          ; Manufacturer
1599 2C03      +1  998          DB      (String2-String1),3 ; Length, Type
159B 55005300 +1  999          DB      "U",0,"S",0,"B",0," ",0,"D",0,"e",0,"s",0,"i",0,"g",0,"n",0," ",0
159F 42002000
15A3 44006500
15A7 73006900
15AB 67006E00
15AF 2000
15B1 42007900 +1 1000          DB      "B",0,"y",0," ",0,"E",0,"x",0,"a",0,"m",0,"p",0,"l",0,"e",0
15B5 20004500
15B9 78006100
15BD 6D007000
15C1 6C006500
15C5          +1 1001          String2:          ; Product Name
15C5 2403      +1 1002          DB      (EndOfDescriptors-String2),3
15C7 4B006500 +1 1003          DB      "K",0,"e",0,"y",0,"b",0,"o",0,"a",0,"r",0,"d",0," ",0
15CB 79006200
15CF 6F006100
15D3 72006400
15D7 2000
15D9 45006D00 +1 1004          DB      "E",0,"m",0,"u",0,"l",0,"a",0,"t",0,"o",0,"r",0
15DD 75006C00
15E1 61007400
15E5 6F007200
15E9          +1 1005          EndOfDescriptors:
15E9 00        +1 1006          DB      0           ; Backstop for String Descriptors
   +1 1007
   +1 1008
   +1 1009
   1010
   1011
   1012
   1013          END

```