

MACRO ASSEMBLER A51 V6.10

OBJECT MODULE PLACED IN .\LCP.OBJ

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

```

LOC  OBJ          LINE    SOURCE
                                1      NAME      LightingControlPanel
                                2
                                3      ;$include (Declare.A51)
+1    4      ; This module declares the variables and constants used in the examples
+1    5      ; It is common to all of the examples
+1    6      ;
+1    7      ; Declare Special Function Registers used
0088  +1    8      TimerControl    DATA    088H
0089  +1    9      TimerMode      DATA    089H
008C  +1   10     Timer0High     DATA    08CH
00A8  +1   11     EI              DATA    0A8H
00E8  +1   12     EIE            DATA    0E8H      ; EZ-USB specific
0091  +1   13     EXIF           DATA    091H      ; EZ-USB specific
00D8  +1   14     EICON          DATA    0D8H      ; EZ-USB specific
0092  +1   15     PageReg       DATA    092H      ; EZ-USB specific, used with MOVX @Ri
0086  +1   16     DPS            DATA    086H      ; EZ-USB specific, used with dual data pointers
+1   17     ;
+1   18     ; "External" memory locations used, EZ-USB specific
+1   19     ; Note that most of these variables are in Page 7FH
7FE8  +1   20     SETUPDAT      EQU     07FE8H
7FD4  +1   21     SUDPTR       EQU     07FD4H
7FB4  +1   22     EP0Control    EQU     07FB4H
7F00  +1   23     EP0InBuffer   EQU     07F00H
7EC0  +1   24     EP0OutBuffer  EQU     07EC0H      ; Not in Page 7FH
7E80  +1   25     EP1InBuffer   EQU     07E80H      ; Not in Page 7FH
7FB5  +1   26     IN0ByteCount  EQU     07FB5H
7FC5  +1   27     Out0ByteCount EQU     07FC5H
7FB7  +1   28     IN1ByteCount  EQU     07FB7H
7FAC  +1   29     IN07IEN        EQU     07FACH
7FA9  +1   30     IN07IRQ       EQU     07FA9H
7FAD  +1   31     OUT07IEN      EQU     07FADH
7FAA  +1   32     OUT07IRQ       EQU     07FAAH
7FAE  +1   33     USBIEN        EQU     07FAEH
7FAB  +1   34     USBIRQ        EQU     07FABH
7FD6  +1   35     USBControl    EQU     07FD6H
7FA6  +1   36     I2CData          EQU     07FA6H
7FA5  +1   37     I2CControl     EQU     07FA5H
7F93  +1   38     PortA_Config   EQU     07F93H
7F94  +1   39     PortB_Config   EQU     07F94H
7F95  +1   40     PortC_Config   EQU     07F95H
7F96  +1   41     PortA_OUT     EQU     07F96H
7F97  +1   42     PortB_OUT     EQU     07F97H
7F98  +1   43     PortC_OUT     EQU     07F98H
7F99  +1   44     PortA_PINS     EQU     07F99H
7F9A  +1   45     PortB_PINS     EQU     07F9AH
7F9B  +1   46     PortC_PINS     EQU     07F9BH
7F9C  +1   47     PortA_OE       EQU     07F9CH
7F9D  +1   48     PortB_OE       EQU     07F9DH
7F9E  +1   49     PortC_OE       EQU     07F9EH
+1   50     ;
+1   51     ; Byte Variables
+1   52
----  +1   53     DSEG            AT 20H
0020  +1   54     FLAGS:        DS            1      ; This register is bit-addressable
+1   55     ; Bit Variables
0000  +1   56     Configured     EQU     FLAGS.0 ; Is this device configured
0001  +1   57     STALL          EQU     FLAGS.1 ; Need to STALL endpoint 0
0002  +1   58     SendData      EQU     FLAGS.2 ; Need to send data to PC Host

```

```
0003      +1 59      IsDescriptor    EQU      FLAGS.3 ; Enable a shortcut reply
          +1 60      ;
0021      +1 61      MonitorSpace: DS      1FH      ; Used by Dscope
0040      +1 62      Temp:          DS      1          ; A temporary working register
0041      +1 63      Idle_Time:     DS      1          ; The time the PC host wants us to wait
0042      +1 64      Expired_Time:  DS      1          ; A downcounter for timed Reports
0043      +1 65      ReplyBuffer:   DS      3          ; First byte is Count
0046      +1 66      CurrentConfiguration:
0046      +1 67                  DS      1          ; Some examples support > 1 configurations
          +1 68      ;
          +1 69      ; Declare the specific variables used by each of the examples
0047      +1 70      Overlay      EQU      $
0047      +1 71      Old_Buttons:  DS      1          ; Used by BAL: stores current button position
0048      +1 72      LEDstrobe:    DS      1          ; Used by BAL: strobe one LED on at a time
0049      +1 73      LEDvalue:     DS      1          ; Used by BAL: stores current LED value
004A      +1 74      Msec_Counter: DS      1          ; Used by BAL: counts up to 4 msec
          +1 75      ;
0047      +1 76                  ORG Overlay      ; Overlay the variables (only one set in use at any one
tim
          e)
0047      +1 77      I2CDataByte:  DS      1          ; Used by I2C: keep a local copy of data read from I2C
bus
          +1 78      ;
0047      +1 79                  ORG Overlay
0047      +1 80      LightValues:  DS      6          ; Used by LP: local buffer for light brightness
004D      +1 81      WorkingValues: DS      6          ; Used by LP: counted down each half cycle
0053      +1 82      Mask:         DS      1          ; Used by LP: TurnON mask for Triacs
0004      +1 83      LastCycle     EQU      FLAGS.4 ; Used by LP: Tracks Positive & Negative Mains half
cycles
          +1 84      ;
0047      +1 85                  ORG Overlay
0047      +1 86      CurrentPosition: DS      1          ; Used by Stepper: motor has 16 stable positions
0048      +1 87      MotorControl:  DS      3          ; Used by Stepper: direction, Low(count) and
High(count)
          +1 88      ;
0047      +1 89                  ORG Overlay
0047      +1 90      LimitValues:  DS      12         ; Used by Temps: local buffer for limits
          +1 91      ;
0047      +1 92                  ORG Overlay
0047      +1 93      ButtonsValue: DS      1          ; Used by RB: buttons are read each full scan
0048      +1 94      DisplayPosition: DS      1          ; Used by RB: holds current display position
0049      +1 95      LEDBuffer:    DS      42         ; Used by RB: local buffer for reader board
          +1 96      ;
          +1 97      ;
          +1 98      ;
          +1 99      ;$include (Vectors.A51)
          +1 100     ; This module is specific to the Lighting Control Panel (since it uses Timer 0)
          +1 101     ;.
          +1 102     ; It contains all of the interrupt vector declarations and
          +1 103     ; the first level interrupt servicing (register save, call subroutine,
          +1 104     ; clear interrupt source, restore registers, return)
          +1 105     ; Suspend and Resume are handled totally in this module
          +1 106     ;
          +1 107     ; A Reset sends us to Program space location 0
----- +1 108     CSEG AT 0          ; Code space
          +1 109     USING 0          ; Reset forces Register Bank 0
0000 020351 +1 110     LJMP Reset
          +1 111     ;
          +1 112     ; The interrupt vector table is also located here
          +1 113     ; EZ-USB has two levels of USB interrupts:
          +1 114     ; 1-the main level is described in this table (at ORG 43H)
          +1 115     ; 2-there are 21 sources of USB interrupts and these are described in USB_ISR
          +1 116     ; This means that two levels of acknowledgement and clearing will be required
          +1 117     ; LJMP INT0_ISR      ; Features not used are commented out
000B      +1 118     ORG 0BH
000B 020157 +1 119     LJMP Timer0_ISR
          +1 120     ; ORG 13H
          +1 121     ; LJMP INT1_ISR
          +1 122     ; ORG 1BH
          +1 123     ; LJMP Timer1_ISR
```

```

+1 124 ; ORG 23H
+1 125 ; LJMP UART0_ISR
+1 126 ; ORG 2BH
+1 127 ; LJMP Timer2_ISR
+1 128 ; ORG 33H
+1 129 ; LJMP WakeUp_ISR
+1 130 ; ORG 3BH
+1 131 ; LJMP UART1_ISR
0043 +1 132 ORG 43H
0043 020100 +1 133 LJMP USB_ISR ; Auto Vector will replace byte 45H
+1 134 ; ORG 4BH
+1 135 ; LJMP I2C_ISR
+1 136 ; ORG 53H
+1 137 ; LJMP INT4_ISR
+1 138 ; ORG 5BH
+1 139 ; LJMP INT5_ISR
+1 140 ; ORG 63H
+1 141 ; LJMP INT6_ISR
+1 142
00E0 +1 143 ORG 0E0H ; Keep out of the way of dScope monitor
+1 144 ; If you are not using dScope then this memory hole
+1 145 ; may be used for useful routines.
0100 +1 146 ORG 100H
0100 02013C +1 147 USB_ISR:LJMP SUDAV_ISR
0103 00 +1 148 DB 0 ; Pad entries to 4 bytes
0104 020118 +1 149 LJMP SOF_ISR
0107 00 +1 150 DB 0
0108 020118 +1 151 LJMP SUTOK_ISR
010B 00 +1 152 DB 0
010C 020129 +1 153 LJMP Suspend_ISR
010F 00 +1 154 DB 0
0110 020120 +1 155 LJMP USBReset_ISR
0113 00 +1 156 DB 0
0114 020118 +1 157 LJMP Reserved
0117 00 +1 158 DB 0
+1 159 ; LJMP EP0In_ISR ; Endpoint Interrupts are not used in these examples
+1 160 ; DB 0 ; Comment out features not used
+1 161 ; LJMP EP0Out_ISR
+1 162 ; DB 0
+1 163 ; LJMP EP1In_ISR
+1 164 ; DB 0
+1 165 ; LJMP EP1Out_ISR
+1 166 ; DB 0
+1 167 ; LJMP EP2In_ISR
+1 168 ; DB 0
+1 169 ; LJMP EP2Out_ISR
+1 170 ; DB 0
+1 171 ; LJMP EP3In_ISR
+1 172 ; DB 0
+1 173 ; LJMP EP3Out_ISR
+1 174 ; DB 0
+1 175 ; LJMP EP4In_ISR
+1 176 ; DB 0
+1 177 ; LJMP EP4Out_ISR
+1 178 ; DB 0
+1 179 ; LJMP EP5In_ISR
+1 180 ; DB 0
+1 181 ; LJMP EP5Out_ISR
+1 182 ; DB 0
+1 183 ; LJMP EP6In_ISR
+1 184 ; DB 0
+1 185 ; LJMP EP6Out_ISR
+1 186 ; DB 0
+1 187 ; LJMP EP7In_ISR
+1 188 ; DB 0
+1 189 ; LJMP EP7Out_ISR

```

```
+1 190      ; End of Interrupt Vector tables
+1 191
+1 192      ; When a feature is used insert the required interrupt processing here
+1 193      ; The example use only used Endpoints 0 and 1 and also SOF for timing
0118      +1 194      Reserved:
0118      +1 195      INT0_ISR:
0118      +1 196      INT1_ISR:
0118      +1 197      Timer1_ISR:
0118      +1 198      UART0_ISR:
0118      +1 199      Timer2_ISR:
0118      +1 200      UART1_ISR:
0118      +1 201      I2C_ISR:
0118      +1 202      INT4_ISR:
0118      +1 203      INT5_ISR:
0118      +1 204      INT6_ISR:
0118      +1 205      SOF_ISR:
0118      +1 206      SUTOK_ISR:
0118      +1 207      EP0In_ISR:
0118      +1 208      EP0Out_ISR:
0118      +1 209      EP1In_ISR:
0118      +1 210      EP1Out_ISR:
0118      +1 211      EP2In_ISR:
0118      +1 212      EP2Out_ISR:
0118      +1 213      EP3In_ISR:
0118      +1 214      EP3Out_ISR:
0118      +1 215      EP4In_ISR:
0118      +1 216      EP4Out_ISR:
0118      +1 217      EP5In_ISR:
0118      +1 218      EP5Out_ISR:
0118      +1 219      EP6In_ISR:
0118      +1 220      EP6Out_ISR:
0118      +1 221      EP7In_ISR :
0118      +1 222      EP7Out_ISR:
0118      +1 223      Not_Used:                ; Should not get any of these
0118 32    +1 224      RETI
+1 225
0119      +1 226      ClearINT2:                ; Tell the hardware that we're done
0119 E591  +1 227      MOV      A, EXIF
011B C2E4  +1 228      CLR      ACC.4                ; Clear the Interrupt 2 bit
011D F591  +1 229      MOV      EXIF, A
011F 22    +1 230      RET
+1 231
0120      +1 232      USBReset_ISR:            ; Bus has been Reset, move to DEFAULT state
0120 C0E0  +1 233      PUSH     ACC
0122 C200  +1 234      CLR      Configured
0124 3119  +1 235      CALL    ClearINT2
+1 236      ; No need to clear source of interrupt
0126 D0E0  +1 237      POP      ACC
0128 32    +1 238      RETI
+1 239
0129      +1 240      Suspend_ISR:              ; SIE detected an Idle bus
0129 C0E0  +1 241      PUSH     ACC
012B E587  +1 242      MOV      A, PCON
012D 4401  +1 243      ORL      A, #1
012F F587  +1 244      MOV      PCON, A                ; Go to sleep!
0131 00    +1 245      NOP
0132 00    +1 246      NOP                ; Wake up here due to a USBResume
0133 00    +1 247      NOP
0134 3119  +1 248      CALL    ClearINT2
0136 D0E0  +1 249      POP      ACC
0138 32    +1 250      RETI
+1 251
0139      +1 252      WakeUp_ISR:                ; Not using external WAKEUP in these examples
+1 253      ; So this must be due to a USBResume
0139 C2DC  +1 254      CLR      EICON.4                ; Clear the wakeup interrupt source
013B 32    +1 255      RETI
```

```
+1 256
013C +1 257 SUDAV_ISR: ; A Setup packet has been received
013C C0D0 +1 258 PUSH PSW ; Save Registers before the service routine
013E C0E0 +1 259 PUSH ACC
0140 C082 +1 260 PUSH DPL
0142 C083 +1 261 PUSH DPH
0144 3163 +1 262 CALL ServiceSetupPacket
0146 3119 +1 263 CALL ClearINT2
+1 264 ; Clear the source of the interrupt
0148 7401 +1 265 MOV A, #00000001b
014A 907FAB +1 266 MOV DPTR, #USBIRQ
014D F0 +1 267 MOVX @DPTR, A
014E D083 +1 268 ExitISR:POP DPH ; Restore Registers
0150 D082 +1 269 POP DPL
0152 D0E0 +1 270 POP ACC
0154 D0D0 +1 271 POP PSW
0156 32 +1 272 RETI
+1 273
0157 +1 274 Timer0_ISR:
0157 C0D0 +1 275 PUSH PSW ; Save Registers before the service routine
0159 C0E0 +1 276 PUSH ACC
015B C082 +1 277 PUSH DPL
015D C083 +1 278 PUSH DPH
015F 71C5 +1 279 CALL ServiceTimerRoutine
+1 280 ; Source of the interrupt cleared automatically
0161 80EB +1 281 JMP ExitISR
+1 282
283
284 ;$include (USB_INT.A51)
+1 285 ; This module is common to all of the examples.
+1 286 ; It services USB Requests from the SIE.
+1 287 ; Interpretation of the Output Reports is handled by MAIN
+1 288 ;
---- +1 289 CSEG
0163 +1 290 ServiceSetupPacket:
0163 907FE8 +1 291 MOV DPTR, #SETUPDAT ; Point to Setup Packet data
0166 E0 +1 292 MOVX A, @DPTR ; Get the RequestType
0167 A2E7 +1 293 MOV C, ACC.7 ; Bit 7 = 1 means IO device needs to send data
to P
C Host
0169 9202 +1 294 MOV SendData, C
016B 545C +1 295 ANL A, #01011100b ; IF RequestType[6.4.3.2] = 1 THEN goto
BadRequest
016D 7050 +1 296 JNZ BadRequest
016F E0 +1 297 MOVX A, @DPTR ; IF RequestType[1&0] = 1 THEN goto BadRequest
0170 A2E0 +1 298 MOV C, ACC.0
0172 82E1 +1 299 ANL C, ACC.1
0174 4049 +1 300 JC BadRequest
0176 30E502 +1 301 JNB ACC.5, NotB5 ; IF RequestType[5] = 1 THEN RequestType[1,0] =
[1,
1]
0179 7403 +1 302 MOV A, #00000011b
017B 5403 +1 303 NotB5: ANL A, #00000011b ; Set CommandIndex[5,4] = RequestType[1,0]
017D C4 +1 304 SWAP A
017E F540 +1 305 MOV Temp, A ; Save HI nibble of CommandIndex
+1 306 ; Set CommandIndex[3,0] = Request[3,0]
0180 A3 +1 307 INC DPTR ; Point to Request
0181 E0 +1 308 MOVX A, @DPTR
0182 540F +1 309 ANL A, #00001111b ; Only 13 are defined today, handle in table
0184 4540 +1 310 ORL A, Temp
0186 31CE +1 311 CALL CorrectSubroutine ; goto CommandTable(CommandIndex)
+1 312 ; Returns STALL=1 if a stall is required
0188 200134 +1 313 JB STALL, BadRequest
018B 300218 +1 314 JNB SendData, HandShake
018E 200320 +1 315 JB IsDescriptor, LoadSUDPTR; EZ-USB has a short cut for descriptors
+1 316 ; Send data in ReplyBuffer
0191 907F02 +1 317 MOV DPTR, #EP0InBuffer+2
0194 7846 +1 318 MOV R0, #ReplyBuffer+3
0196 754003 +1 319 MOV Temp, #3 ; Copy maximum byte count
```

```
0199 E6      +1 320      CopyRB: MOV      A, @R0
019A F0      +1 321              MOVX     @DPTR, A
019B 1582    +1 322              DEC      DPL
019D 18      +1 323              DEC      R0
019E D540F8  +1 324              DJNZ    Temp, CopyRB
01A1 E6      +1 325              MOV      A, @R0                ; Get real byte count
01A2         +1 326      SendEP0InBuffer:
01A2 907FB5  +1 327              MOV      DPTR, #In0ByteCount
01A5         +1 328      StartXfer:
01A5 F0      +1 329              MOVX     @DPTR, A                ; This write initiates the transfer
01A6         +1 330      HandShake:                ; Handshake with host
01A6 754002  +1 331              MOV      Temp, #00000010b      ; Set HSNACK to tell the SIE that we're done
01A9         +1 332      SetEP0Control:
01A9 907FB4  +1 333              MOV      DPTR, #EP0Control
01AC E0      +1 334              MOVX     A, @DPTR
01AD 4540    +1 335              ORL     A, Temp
01AF F0      +1 336              MOVX     @DPTR, A
01B0 22      +1 337              RET
01B1         +1 338      LoadSUDPTR:                ; Send the data pointed to by DPTR
01B1 858240  +1 339              MOV      Temp, DPL
01B4 E583    +1 340              MOV      A, DPH
01B6 907FD4  +1 341              MOV      DPTR, #SUDPTR
01B9 F0      +1 342              MOVX     @DPTR, A
01BA E540    +1 343              MOV      A, Temp
01BC A3      +1 344              INC      DPTR
01BD 80E6    +1 345              JMP      StartXfer
01BF         +1 346      BadRequest:                ; Invalid Request was received
01BF 754003  +1 347              MOV      Temp, #00000011b     ; Set EPOSTALL and HSNACK
01C2 80E5    +1 348              JMP      SetEP0Control
01C4         +1 350      NextDPTR:                ; Returns (DPTR + byte DPTR is pointing to)
01C4 E0      +1 351              MOVX     A, @DPTR
01C5         +1 352      BumpDPTR:                ; Returns (DPTR + ACC)
01C5 2582    +1 353              ADD     A, DPL
01C7 F582    +1 354              MOV      DPL, A
01C9 5002    +1 355              JNC     Skip
01CB 0583    +1 356              INC     DPH                ; Need 16 bit arithmetic here
01CD 22      +1 357      Skip:  RET
01CE         +1 358
01CE         +1 359      CorrectSubroutine:        ; Jump to the subroutine that DPTR is pointing
to
01CE 9001F3  +1 360              MOV      DPTR, #CommandTable
01D1 31C5    +1 361              CALL    BumpDPTR                ; Point to entry
01D3 E0      +1 362              MOVX     A, @DPTR                ; Get the offset
01D4 9001F3  +1 363              MOV      DPTR, #CommandTable
01D7 31C5    +1 364              CALL    BumpDPTR                ; Get the routine address
01D9 C082    +1 365              PUSH    DPL                ; Create a RETURN address on stack
01DB C083    +1 366              PUSH    DPH                ; Note: JMP @A+DPTR not used since A, DPTR
needed
01DD 7845    +1 367              MOV      R0, #ReplyBuffer+2
01DF E4      +1 368              CLR     A
01E0 F6      +1 369              MOV     @R0, A                ; Clear ReplyBuffer
01E1 18      +1 370              DEC     R0
01E2 F6      +1 371              MOV     @R0, A
01E3 18      +1 372              DEC     R0
01E4 7601    +1 373              MOV     @R0, #1                ; Default non-descriptor reply
01E6 907FEA  +1 374              MOV     DPTR, #SETUPDAT+2      ; Point to LOW(wValue)
01E9 E0      +1 375              MOVX    A, @DPTR                ; Many of the routines need these
01EA F5F0    +1 376              MOV     B, A                ; LOW(wValue) in B
01EC A3      +1 377              INC     DPTR
01ED E0      +1 378              MOVX    A, @DPTR                ; HIGH(wValue) in A
01EE C201    +1 379              CLR     STALL
01F0 C203    +1 380              CLR     IsDescriptor
01F2 22      +1 381              RET                ; Go to service routine
01F2         +1 382
01F2         +1 383      ; Since the table only contains byte offsets, it is important that all these routines
are
01F2         +1 384      ; within one page (100H) of CommandTable
01F2         +1 385      ;
```

```
01F3          +1 386      CommandTable:
                +1 387      ; First 16 commands are for the Device
01F3 6C       +1 388      DB Device_Get_Status - CommandTable
01F4 40       +1 389      DB Device_Clear_Feature - CommandTable
01F5 40       +1 390      DB Invalid - CommandTable
01F6 40       +1 391      DB Device_Set_Feature - CommandTable
01F7 40       +1 392      DB Invalid - CommandTable
01F8 40       +1 393      DB Invalid - CommandTable                ; SIE implements Device_Set_Address
01F9 80       +1 394      DB Get_Descriptor - CommandTable
01FA 40       +1 395      DB Set_Descriptor - CommandTable
01FB 69       +1 396      DB Get_Configuration - CommandTable
01FC 73       +1 397      DB Set_Configuration - CommandTable
01FD 40       +1 398      DB Invalid - CommandTable
01FE 40       +1 399      DB Invalid - CommandTable
01FF 40       +1 400      DB Invalid - CommandTable
0200 40       +1 401      DB Invalid - CommandTable
0201 40       +1 402      DB Invalid - CommandTable
0202 40       +1 403      DB Invalid - CommandTable
                +1 404      ; Next 16 commands are for the Interface
0203 70       +1 405      DB Interface_Get_Status - CommandTable
0204 40       +1 406      DB Interface_Clear_Feature - CommandTable
0205 40       +1 407      DB Invalid - CommandTable
0206 40       +1 408      DB Interface_Set_Feature - CommandTable
0207 40       +1 409      DB Invalid - CommandTable
0208 40       +1 410      DB Invalid - CommandTable
0209 A4       +1 411      DB Get_Class_Descriptor - CommandTable
020A 40       +1 412      DB Set_Class_Descriptor - CommandTable
020B 40       +1 413      DB Invalid - CommandTable
020C 40       +1 414      DB Invalid - CommandTable
020D 40       +1 415      DB Get_Interface - CommandTable
020E 40       +1 416      DB Set_Interface - CommandTable
020F 40       +1 417      DB Invalid - CommandTable
0210 40       +1 418      DB Invalid - CommandTable
0211 40       +1 419      DB Invalid - CommandTable
0212 40       +1 420      DB Invalid - CommandTable
                +1 421      ; Next 16 commands are for the Endpoint
0213 70       +1 422      DB Endpoint_Get_Status - CommandTable
0214 42       +1 423      DB Endpoint_Clear_Feature - CommandTable
0215 40       +1 424      DB Invalid - CommandTable
0216 40       +1 425      DB Endpoint_Set_Feature - CommandTable
0217 40       +1 426      DB Invalid - CommandTable
0218 40       +1 427      DB Invalid - CommandTable
0219 40       +1 428      DB Invalid - CommandTable
021A 40       +1 429      DB Invalid - CommandTable
021B 40       +1 430      DB Invalid - CommandTable
021C 40       +1 431      DB Invalid - CommandTable
021D 40       +1 432      DB Invalid - CommandTable
021E 40       +1 433      DB Invalid - CommandTable
021F 40       +1 434      DB Endpoint_Sync_Frame - CommandTable
0220 40       +1 435      DB Invalid - CommandTable
0221 40       +1 436      DB Invalid - CommandTable
0222 40       +1 437      DB Invalid - CommandTable
                +1 438      ; Next 16 commands are Class Requests
0223 40       +1 439      DB Invalid - CommandTable
0224 55       +1 440      DB Get_Report - CommandTable
0225 62       +1 441      DB Get_Idle - CommandTable
0226 40       +1 442      DB Get_Protocol - CommandTable
0227 40       +1 443      DB Invalid - CommandTable
0228 40       +1 444      DB Invalid - CommandTable
0229 40       +1 445      DB Invalid - CommandTable
022A 40       +1 446      DB Invalid - CommandTable
022B 40       +1 447      DB Invalid - CommandTable
022C 43       +1 448      DB Set_Report - CommandTable
022D 5C       +1 449      DB Set_Idle - CommandTable
022E 40       +1 450      DB Set_Protocol - CommandTable
022F 40       +1 451      DB Invalid - CommandTable
```

```
0230 40      +1 452          DB Invalid - CommandTable
0231 40      +1 453          DB Invalid - CommandTable
0232 40      +1 454          DB Invalid - CommandTable
          +1 455          ;
          +1 456          ; Many requests are INVALID for this example
0233          +1 457      Get_Protocol:          ; We are not a Boot device
0233          +1 458      Set_Protocol:          ; We are not a Boot device
0233          +1 459      Set_Descriptor:        ; Our Descriptors are static
0233          +1 460      Set_Class_Descriptor:   ; Our Descriptors are static
0233          +1 461      Set_Interface:         ; We only have one Interface
0233          +1 462      Get_Interface:        ; We do not have an Alternate setting
0233          +1 463      Device_Set_Feature:    ; We have no features that can be set or cleared
0233          +1 464      Interface_Set_Feature: ; We have no features that can be set or cleared
0233          +1 465      Endpoint_Set_Feature:  ; We have no features that can be set or cleared
0233          +1 466      Device_Clear_Feature:  ; We have no features that can be set or cleared
0233          +1 467      Interface_Clear_Feature; We have no features that can be set or cleared
0233          +1 468      Endpoint_Sync_Frame:   ; We are not an Isonchronous device
          +1 469
0233          +1 470      Invalid:              ; Invalid Request made, STALL the Endpoint
0233 D201    +1 471          SETB     STALL
          +1 472          ;
0235          +1 473      Endpoint_Clear_Feature; We have no features that can be set or cleared
          +1 474          ;
0235 22      +1 475      Reply:  RET
          +1 476
0236          +1 477      Set_Report:          ; Host wants to sent us a Report.
          +1 478      ; The ONLY case in this example where host sends data to us
0236 3000FA  +1 479          JNB     Configured, Invalid ; Need to be Configured to do this command
0239 907FC5  +1 480          MOV     DPTR, #Out0ByteCount ; Enable EP0OutBuffer to receive data
023C F0      +1 481          MOVX    @DPTR, A          ; Any value will do
023D 907FAA  +1 482          MOV     DPTR, #OUT07IRQ    ; Wait for valid data in EP0OutBuffer
0240 E0      +1 483      Wait4D: MOVX    A, @DPTR
0241 5401    +1 484          ANL     A, #00000001b
0243 60FB    +1 485          JZ      Wait4D
0245 F0      +1 486          MOVX    @DPTR, A          ; Clear the interrupt
0246 61B7    +1 487          JMP     ProcessOutputReport ; RETurn via this subroutine
0248          +1 488      Get_Report:          ; Host wants a Report
0248 3000E8  +1 489          JNB     Configured, Invalid ; Need to be Configured to do this command
024B 08      +1 490          INC     R0              ; Point to ReplyBuffer(1)
024C 7618    +1 491          MOV     @R0, #18H        ; Reply with a recognizable (arbitrary) value
024E 22      +1 492          RET
024F          +1 493      Set_Idle:          ; Host wants to tell us how often we should
talk
024F 3000E1  +1 494          JNB     Configured, Invalid ; Need to be Configured to do this command
0252 F541    +1 495          MOV     Idle_Time, A
0254 22      +1 496          RET              ; Handshake with host
0255          +1 497      Get_Idle:          ; Host must have forgotten what he told us to
do
0255 3000DB  +1 498          JNB     Configured, Invalid ; Need to be Configured to do this command
0258 08      +1 499          INC     R0              ; Point to ReplyBuffer(1)
0259 A641    +1 500          MOV     @R0, Idle_Time
025B 22      +1 501          RET
025C          +1 502      Get_Configuration:    ; Need to return 0 or 1
025C 300004  +1 503          JNB     Configured, Configuration0
025F          +1 504      Configuration1:      ; Same bit pattern as Device_Get_Status
025F          +1 505      Device_Get_Status:   ; Only two bits of Device Status are defined
025F 08      +1 506          INC     R0              ; Point to ReplyBuffer(1)
0260 7601    +1 507          MOV     @R0, #1          ; Bit 1=Remote Wakeup(=0), Bit 0=Self
Powered(=1)
0262 22      +1 508          RET
0263          +1 509      Configuration0:      ; Same bit pattern as Interface_Get_Status
0263          +1 510      Interface_Get_Status: ; Interface Status is currently defined as 0
0263          +1 511      Endpoint_Get_Status:
0263 7602    +1 512          MOV     @R0, #2
0265 22      +1 513          RET
0266          +1 514      Set_Configuration:    ; Valid values are 0 and 1
0266 E5F0    +1 515          MOV     A, B          ; Get LOW(wValue)
0268 6006    +1 516          JZ      Deconfigured
026A 14      +1 517          DEC     A
```

```
026B 70C6      +1  518          JNZ      Invalid
026D D200      +1  519          SETB    Configured
026F 22        +1  520          RET
0270          +1  521      Deconfigured:
0270 C200      +1  522          CLR      Configured
0272 22        +1  523          RET
0273          +1  524      Get_Descriptor:
0273 D203      +1  525          SETB    IsDescriptor      ; Host wants to know who/what we are
0275 14        +1  526          DEC      A                ; Valid Values are 1, 2 and 3
0276 9002BD    +1  527          MOV     DPTR, #DeviceDescriptor
0279 60BA      +1  528          JZ      Reply
027B 14        +1  529          DEC      A
027C 9002CF    +1  530          MOV     DPTR, #ConfigurationDescriptor
027F 60B4      +1  531          JZ      Reply
0281 14        +1  532          DEC      A
0282 70AF      +1  533          JNZ     Invalid
+1  534          ; Request is for a String Descriptor
0284 900301    +1  535          MOV     DPTR, #String0      ; Point to String 0
0287 E5F0      +1  536          MOV     A, B                ; Get String Index
0289          +1  537      NextString:
0289 601E      +1  538          JZ      FixUpthenReply
028B F540      +1  539          MOV     Temp, A            ; Save String Index
028D 31C4      +1  540          CALL   NextDPTR
028F E0        +1  541          MOVX   A, @DPTR            ; Get the String Length (= 0 means we're at
Backsto
p)
0290 60A1      +1  542          JZ      Invalid            ; Asked for a string I don't have
0292 E540      +1  543          MOV     A, Temp
0294 14        +1  544          DEC      A
0295 80F2      +1  545          JMP     NextString         ; Check if we are there yet
0297          +1  546      Get_Class_Descriptor:
Request
0297 D203      +1  547          SETB    IsDescriptor
0299 C3        +1  548          CLR      C
029A 9421      +1  549          SUBB   A, #21H
029C 9002E1    +1  550          MOV     DPTR, #HIDDescriptor
029F 6094      +1  551          JZ      Reply
02A1 14        +1  552          DEC      A
02A2 9002EA    +1  553          MOV     DPTR, #ReportDescriptor
02A5 608E      +1  554          JZ      Reply
+1  555          ; DEC      A                ; This example does not use Physical
Descriptors
+1  556          ; JZ      Send_Physical_Descriptor
02A7 808A      +1  557          JMP     Invalid
+1  558          ;
+1  559          ; Error check: this MUST be on within a page of CommandTable
00B6          +1  560      WithinSamePage EQU $ - CommandTable
+1  561          ;
02A9          +1  562      FixUpthenReply:
+1  563          ; EZ-USB Rev D has a String Descriptor bug
02A9 E0        +1  564          MOVX   A, @DPTR            ; Get the string length
02AA FF        +1  565          MOV     R7, A                ; Save counter
02AB F5F0      +1  566          MOV     B, A
02AD 7800      +1  567          MOV     R0, #LOW(EP0InBuffer) ; PageReg = 7FH = HIGH(EP0InBuffer)
02AF F2        +1  568      CopySD: MOVX   @R0, A
02B0 08        +1  569          INC     R0
02B1 A3        +1  570          INC     DPTR
02B2 E0        +1  571          MOVX   A, @DPTR
02B3 DFFA      +1  572          DJNZ   R7, CopySD
+1  573          ; Fixup complete, get back to the program flow
02B5 D0E0      +1  574          POP     ACC                ; Get rid of the return address
02B7 D0E0      +1  575          POP     ACC
02B9 E5F0      +1  576          MOV     A, B                ; Retrieve byte count
02BB 21A2      +1  577          JMP     SendEP0InBuffer
578
579          ;$include (DTables.A51)
+1  580          ; This module declares the descriptors
+1  581          ;
+1  582          ; This example has one Device Descriptor with:
```

```

+1 583 ; One Configuration - single IN port and single OUT port
+1 584 ; One Interface - there is only one method of accessing the ports
+1 585 ; One HID Descriptor - to make PC host software simpler
+1 586 ; No Endpoint Descriptors - HID Output Reports use EP0
+1 587 ; One Report Descriptor - six bytes OUT
+1 588 ; Multiple Sting Descriptors - to aid the user
+1 589 ;
----
+1 590 CSEG
02BD +1 591 DeviceDescriptor:
02BD 1201 +1 592 DB 18, 1 ; Length, Type
02BF 0101 +1 593 DW 101H ; USB Rev 1.1
02C1 000000 +1 594 DB 0, 0, 0 ; Class, Subclass and Protocol
02C4 40 +1 595 DB 64 ; EP0 size
02C5 4242 +1 596 DW 4242H, 1, 1 ; Vendor ID, Product ID and Version
02C7 0001
02C9 0001
02CB 010200 +1 597 DB 1, 2, 0 ; Manufacturer, Product & Serial# Names
02CE 01 +1 598 DB 1 ; #Configs
02CF +1 599 ConfigurationDescriptor:
02CF 0902 +1 600 DB 9, 2 ; Length, Type
02D1 1B00 +1 601 DB LOW(ConfigLength), HIGH(ConfigLength)
02D3 010100 +1 602 DB 1, 1, 0 ; #Interfaces, Configuration#, Config. Name
02D6 80 +1 603 DB 10000000b ; Attributes = Bus Powered
02D7 32 +1 604 DB 50 ; Max. Power is 50x2 = 100mA
02D8 +1 605 InterfaceDescriptor:
02D8 0904 +1 606 DB 9, 4 ; Length, Type
02DA 000000 +1 607 DB 0, 0, 0 ; No alternate setting, HID OUTPUT uses EP0
02DD 03 +1 608 DB 3 ; Class = Human Interface Device
02DE 0000 +1 609 DB 0, 0 ; Subclass and Protocol
02E0 00 +1 610 DB 0 ; Interface Name
02E1 +1 611 HIDDescriptor:
02E1 0921 +1 612 DB 9, 21H ; Length, Type
02E3 0001 +1 613 DB 0, 1 ; HID Class Specification compliance
02E5 00 +1 614 DB 0 ; Country localization (=none)
02E6 01 +1 615 DB 1 ; Number of descriptors to follow
02E7 22 +1 616 DB 22H ; And it's a Report descriptor
02E8 1700 +1 617 DB LOW(ReportLength), HIGH(ReportLength)
001B +1 618 ConfigLength EQU $ - ConfigurationDescriptor
+1 619
02EA +1 620 ReportDescriptor: ; Generated with HID Tool, copied to here
02EA 0600FF +1 621 DB 6, 0, 0FFH ; Usage_Page (Vendor Defined)
02ED 0901 +1 622 DB 9, 1 ; Usage (I/O Device)
02EF A101 +1 623 DB 0A1H, 1 ; Collection (Application)
02F1 1901 +1 624 DB 19H, 1 ; Usage_Minimum
02F3 2902 +1 625 DB 29H, 2 ; Usage_Maximum
02F5 1500 +1 626 DB 15H, 0 ; Logical_Minimum (0)
02F7 26FF00 +1 627 DB 26H, 255, 0 ; Logical_Maximum (255)
02FA 7508 +1 628 DB 75H, 8 ; Report_Size (8)
02FC 9506 +1 629 DB 95H, 6 ; Report_Count (6) = Lighting Values
02FE 9102 +1 630 DB 91H, 2 ; Output (Data,Var,Abs)
0300 C0 +1 631 DB 0C0H ; End_Collection
0017 +1 632 ReportLength EQU $-ReportDescriptor
+1 633
0301 +1 634 String0: ; Declare the UNICODE strings
0301 04030904 +1 635 DB 4, 3, 9, 4 ; Only English language strings supported
0305 +1 636 String1: ; Manufacturer
0305 2C03 +1 637 DB (String2-String1),3 ; Length, Type
0307 55005300 +1 638 DB "U",0,"S",0,"B",0," ",0,"D",0,"e",0,"s",0,"i",0,"g",0,"n",0," ",0
030B 42002000
030F 44006500
0313 73006900
0317 67006E00
031B 2000
031D 42007900 +1 639 DB "B",0,"y",0," ",0,"E",0,"x",0,"a",0,"m",0,"p",0,"l",0,"e",0
0321 20004500
0325 78006100

```

```
0329 6D007000
032D 6C006500
0331          +1 640      String2:                ; Product Name
0331 1E03      +1 641      DB          (EndOfDescriptors-String2),3
0333 4C006900 +1 642      DB          "L",0,"i",0,"g",0,"h",0,"t",0,"i",0,"n",0,"g",0
0337 67006800
033B 74006900
033F 6E006700
0343 20005000 +1 643      DB          " ",0,"P",0,"a",0,"n",0,"e",0,"l",0
0347 61006E00
034B 65006C00
034F          +1 644      EndOfDescriptors:
034F 0000      +1 645      DW          0                ; Backstop for String Descriptors
+1 646
+1 647
+1 648
+1 649
+1 650      ;$include (Main.A51)
+1 651      ; This module initializes the microcontroller then executes MAIN forever
+1 652      ;
+1 653
0351          +1 654      Reset:
0351 7581EB    +1 655      MOV          SP, #235                ; Initialize the Stack at top of internal
memory
0354 75927F    +1 656      MOV          PageReg, #7FH          ; Needed to use MOVX @Ri
+1 657
0357 78D6      +1 658      MOV          R0, #LOW(USBControl) ; Simulate a disconnect
0359 E2        +1 659      MOVX         A, @R0
035A 54F3      +1 660      ANL          A, #11110011b        ; Clear DISCON, DISCOE
035C F2        +1 661      MOVX         @R0, A
035D 71A6      +1 662      CALL         Wait100msec          ; Give the host time to react
035F E2        +1 663      MOVX         A, @R0
0360 4406      +1 664      ORL          A, #00000110b        ; Set DISCOE to enable pullup resistor
0362 F2        +1 665      MOVX         @R0, A
0363 E4        +1 666      CLR          A
0364 F520      +1 667      MOV          FLAGS, A             ; Start in Default state
0366          +1 668      InitializeIOSystem:              ; This example uses PortA an OUT and
+1 669      ; the lower 4 bits of PortC as IN
+1 670      ; Assume a pre-existing configuration (ie Dscope)
0366 7893      +1 671      MOV          R0, #LOW(PortA_Config) ; PageReg = 7F = HIGH(PortA_Config)
0368 E4        +1 672      CLR          A
0369 F2        +1 673      MOVX         @R0, A
036A 799C      +1 674      MOV          R1, #LOW(PortA_OE)
036C F4        +1 675      CPL          A
036D F3        +1 676      MOVX         @R1, A
036E 7895      +1 677      MOV          R0, #LOW(PortC_Config) ; PageReg = 7F = HIGH(PortC_Config)
0370 799E      +1 678      MOV          R1, #LOW(PortC_OE)
0372 E2        +1 679      MOVX         A, @R0
0373 54F0      +1 680      ANL          A, #0F0H
0375 F2        +1 681      MOVX         @R0, A
0376 E3        +1 682      MOVX         A, @R1
0377 54F0      +1 683      ANL          A, #0F0H
0379 F3        +1 684      MOVX         @R1, A
+1 685      ; Enable PortC_Bits[3:0] for Input
+1 686      ; Need to initialize Timer 0 to generate 40 microsecond interrupts
037A 758C50    +1 686      MOV          Timer0High, #80      ; Reload Value
037D E589      +1 687      MOV          A, TimerMode
037F 54F0      +1 688      ANL          A, #0F0H
0381 D2E1      +1 689      SETB         ACC.1
0383 F589      +1 690      MOV          TimerMode, A
0385 E588      +1 691      MOV          A, TimerControl
0387 4430      +1 692      ORL          A, #00110000b        ; Turn on Timer 0
0389 F588      +1 693      MOV          TimerControl, A
+1 694
038B          +1 695      InitializeInterruptSystem:       ; First initialize the USB level
038B 78AC      +1 696      MOV          R0, #LOW(IN07IEN)
038D F2        +1 697      MOVX         @R0, A
038E 08        +1 698      INC          R0
; Disable interrupts from IN Endpoints 0-7
```

```
038F F2      +1 699      MOVX    @R0, A      ; Disable interrupts from OUT Endpoints 0-7
0390 08      +1 700      INC     R0
0391 7401    +1 701      MOV     A, #00000001b
0393 F2      +1 702      MOVX    @R0, A      ; Enable (Resume, Suspend) and SUDAV INTs
0394 08      +1 703      INC     R0
0395 7401    +1 704      MOV     A, #00000001b
0397 F2      +1 705      MOVX    @R0, A      ; Enable Auto Vectoring for USB interrupts
0398 78AA    +1 706      MOV     R0, #LOW(OUT07IRQ)
039A 74FF    +1 707      MOV     A, #0FFH
039C F2      +1 708      MOVX    @R0, A      ; Clear out any pending interrupts
                                +1 709      ; Now enable the main level
039D 75E801  +1 710      MOV     EIE, #00000001b ; Enable INT2 = USB Interrupt (only)
03A0 75A8C2  +1 711      MOV     EI, #11000010b ; Enable interrupt subsystem: Timer 0 overflow
                                +1 712      ; (and Ser1 for Dscope)
                                +1 713
                                +1 714      ; Initialization Complete.
                                +1 715      ;
03A3        +1 716      MAIN:
03A3 00      +1 717      NOP                                ; Not much of a main loop for this example
03A4 80FD    +1 718      JMP     MAIN                        ; All actions are initiated by interrupts
                                +1 719      ; We are a slave, we wait to be told what to do
                                +1 720
03A6        +1 721      Wait100msec:
03A6 754064  +1 722      MOV     Temp, #100
03A9        +1 723      Wait1msec:                          ; A delay loop
03A9 90FB50  +1 724      MOV     DPTR, #-1200
03AC A3      +1 725      More:  INC     DPTR                  ; 3 cycles
03AD E582    +1 726      MOV     A, DPL                      ; + 2
03AF 4583    +1 727      ORL    A, DPH                      ; + 2
03B1 70F9    +1 728      JNZ    More                        ; + 3 = 10 cycles x 1200 = 1msec
03B3 D540F3  +1 729      DJNZ   Temp, Wait1msec
03B6 22      +1 730      RET
                                +1 731
03B7        +1 732      ProcessOutputReport:                ; A Report has just been received
                                +1 733      ; The report is six bytes long
                                +1 734      ; Save the values for the INTERRUPT service routine
03B7 7847    +1 735      MOV     R0, #LightValues           ; Initialize the pointers to be used
03B9 907EC0  +1 736      MOV     DPTR, #EP0OutBuffer       ; Point to the Report
03BC 7F06    +1 737      MOV     R7, #6
03BE E0      +1 738      CopyOR: MOVX    A, @DPTR            ; Retrieve Report Byte 1
03BF F6      +1 739      MOV     @R0, A
03C0 A3      +1 740      INC     DPTR
03C1 08      +1 741      INC     R0
03C2 DFFA    +1 742      DJNZ   R7, CopyOR
03C4 22      +1 743      RET
                                +1 744
03C5        +1 745      CreateInputReport:
                                +1 746      ; Not used in this example
                                +1 747
                                +1 748
                                +1 749
                                +1 750      ;$include (Timer.A51)
                                +1 751      ; This module services the real time interrupt
                                +1 752      ;
                                +1 753      ; Get a Real Time interrupt every 40 microseconds (from Timer 0)
                                +1 754      ;
                                +1 755      ; This routine generates a 40usec pulse which turns on a Triac (via an opto-isolator)
                                +1 756      ; The 10msec LINE half cycle is divided into 250 40usec 'slots'
                                +1 757      ; The position of the TurnON pulse will vary the phase angle of the power control
                                +1 758      ;
                                +1 759      ; The PC Host provides 6 "brightness" values, these are counted down each half-cycle
                                +1 760      ;
                                +1 761      ; The Triacs are on Port A bits [5:0]
                                +1 762      ; A Line zero-cross detector is on Port C bit 0
                                +1 763      ;
03C5        +1 764      ServiceTimerRoutine:
```

```
03C5 907F9B +1 765 MOV DPTR, #PortC_Pins ; First check if we have just changes cycles
03C8 E0 +1 766 MOVX A, @DPTR
03C9 907F96 +1 767 MOV DPTR, #PortA_OUT ; Will need this later
03CC A204 +1 768 MOV C, LastCycle
03CE 30E001 +1 769 JNB ACC.0, PositiveCycle ; Need an XRL C, but we don't have one!
03D1 B3 +1 770 CPL C
03D2 +1 771 PositiveCycle:
03D2 4014 +1 772 JC SameCycle
03D4 +1 773 CycleChange:
03D4 A2E0 +1 774 MOV C, ACC.0 ; Retrieve current cycle
03D6 9204 +1 775 MOV LastCycle, C ; Save it for next time
03D8 7480 +1 776 MOV A, #10000000b ; Ensure all TurnON signals are low
03DA F0 +1 777 MOVX @DPTR, A ; Trigger the 'scope on Bit 7
03DB 7847 +1 778 MOV R0, #LightValues
03DD 794D +1 779 MOV R1, #WorkingValues
03DF 7F06 +1 780 MOV R7, #6
03E1 E6 +1 781 CCLoop: MOV A, @R0 ; Get the Light Values
03E2 F4 +1 782 CPL A ; Since we count down
03E3 F7 +1 783 MOV @R1, A ; Update the Working Values
03E4 08 +1 784 INC R0
03E5 09 +1 785 INC R1
03E6 DFF9 +1 786 DJNZ R7, CCLoop ; Copy all six values
03E8 +1 787 SameCycle: ; Check to see if any counters have expired
03E8 784D +1 788 MOV R0, #WorkingValues
03EA 755300 +1 789 MOV Mask, #0 ; Allow for XCH
03ED 7420 +1 790 MOV A, #00100000b ; Turn on signal for a Triac
03EF +1 791 NextTriac:
03EF C553 +1 792 XCH A, Mask ; Accumulate TurnON signals in A
03F1 8640 +1 793 MOV Temp, @R0
03F3 D54002 +1 794 DJNZ Temp, KeepOFF
03F6 4553 +1 795 ORL A, Mask ; Set a TurnON bit
03F8 A640 +1 796 KeepOFF: MOV @R0, Temp
03FA 08 +1 797 INC R0 ; Ready for next loop
03FB C553 +1 798 XCH A, Mask
03FD 03 +1 799 RR A ; Rotate Mask pattern
03FE 70EF +1 800 JNZ NextTriac
0400 E553 +1 801 MOV A, Mask ; Retrieve TurnON pattern
0402 F0 +1 802 MOVX @DPTR, A ; TurnON triac(s) if required
0403 22 +1 803 RET
+1 804
+1 805
+1 806
+1 807 END
```