

MACRO ASSEMBLER A51 V6.10

OBJECT MODULE PLACED IN .\I2C.OBJ

ASSEMBLER INVOKED BY: C:\Keil\C51\BIN\A51.EXE .\I2C.A51 REGISTERBANK(0) SET(SMALL) DEBUG EP

```

LOC  OBJ          LINE      SOURCE
          1      NAME      I2Cbridge
          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 common to all of the examples.
          +1 101  ; It contains all of the interrupt vector declarations and
          +1 102  ; the first level interrupt servicing (register save, call subroutine,
          +1 103  ; clear interrupt source, restore registers, return)
          +1 104  ; Suspend and Resume are handled totally in this module
          +1 105  ;
          +1 106  ; A Reset sends us to Program space location 0
----      +1 107          CSEG AT 0      ; Code space
          +1 108          USING 0      ; Reset forces Register Bank 0
0000 02035C +1 109          LJMP   Reset
          +1 110  ;
          +1 111  ; The interrupt vector table is also located here
          +1 112  ; EZ-USB has two levels of USB interrupts:
          +1 113  ; 1-the main level is described in this table (at ORG 43H)
          +1 114  ; 2-there are 21 sources of USB interrupts and these are described in USB_ISR
          +1 115  ; This means that two levels of acknowledgement and clearing will be required
          +1 116  ; LJMP   INT0_ISR    ; Features not used are commented out
          +1 117  ; ORG    0BH
          +1 118  ; LJMP   Timer0_ISR
          +1 119  ; ORG    13H
          +1 120  ; LJMP   INT1_ISR
          +1 121  ; ORG    1BH
          +1 122  ; LJMP   Timer1_ISR
          +1 123  ; ORG    23H
```

```
+1 124 ; LJMP UART0_ISR
+1 125 ; ORG 2BH
+1 126 ; LJMP Timer2_ISR
+1 127 ; ORG 33H
+1 128 ; LJMP WakeUp_ISR
+1 129 ; ORG 3BH
+1 130 ; LJMP UART1_ISR
0043 +1 131 ORG 43H
0043 020100 +1 132 LJMP USB_ISR ; Auto Vector will replace byte 45H
+1 133 ; ORG 4BH
+1 134 ; LJMP I2C_ISR
+1 135 ; ORG 53H
+1 136 ; LJMP INT4_ISR
+1 137 ; ORG 5BH
+1 138 ; LJMP INT5_ISR
+1 139 ; ORG 63H
+1 140 ; LJMP INT6_ISR
+1 141
00E0 +1 142 ORG 0E0H ; Keep out of the way of dScope monitor
+1 143 ; If you are not using dScope then this memory hole
+1 144 ; may be used for useful routines.
0100 +1 145 ORG 100H
0100 02013C +1 146 USB_ISR:LJMP SUDAV_ISR
0103 00 +1 147 DB 0 ; Pad entries to 4 bytes
0104 020157 +1 148 LJMP SOF_ISR
0107 00 +1 149 DB 0
0108 020118 +1 150 LJMP SUTOK_ISR
010B 00 +1 151 DB 0
010C 020129 +1 152 LJMP Suspend_ISR
010F 00 +1 153 DB 0
0110 020120 +1 154 LJMP USBReset_ISR
0113 00 +1 155 DB 0
0114 020118 +1 156 LJMP Reserved
0117 00 +1 157 DB 0
+1 158 ; LJMP EP0In_ISR ; Endpoint Interrupts are not used in these examples
+1 159 ; DB 0 ; Comment out features not used
+1 160 ; LJMP EP0Out_ISR
+1 161 ; DB 0
+1 162 ; LJMP EP1In_ISR
+1 163 ; DB 0
+1 164 ; LJMP EP1Out_ISR
+1 165 ; DB 0
+1 166 ; LJMP EP2In_ISR
+1 167 ; DB 0
+1 168 ; LJMP EP2Out_ISR
+1 169 ; DB 0
+1 170 ; LJMP EP3In_ISR
+1 171 ; DB 0
+1 172 ; LJMP EP3Out_ISR
+1 173 ; DB 0
+1 174 ; LJMP EP4In_ISR
+1 175 ; DB 0
+1 176 ; LJMP EP4Out_ISR
+1 177 ; DB 0
+1 178 ; LJMP EP5In_ISR
+1 179 ; DB 0
+1 180 ; LJMP EP5Out_ISR
+1 181 ; DB 0
+1 182 ; LJMP EP6In_ISR
+1 183 ; DB 0
+1 184 ; LJMP EP6Out_ISR
+1 185 ; DB 0
+1 186 ; LJMP EP7In_ISR
+1 187 ; DB 0
+1 188 ; LJMP EP7Out_ISR
+1 189 ; End of Interrupt Vector tables
```

```
+1 190
+1 191 ; When a feature is used insert the required interrupt processing here
+1 192 ; The example use only used Endpoints 0 and 1 and also SOF for timing
0118 +1 193 Reserved:
0118 +1 194 INT0_ISR:
0118 +1 195 Timer0_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 SUTOK_ISR:
0118 +1 206 EP0In_ISR:
0118 +1 207 EP0Out_ISR:
0118 +1 208 EP1In_ISR:
0118 +1 209 EP1Out_ISR:
0118 +1 210 EP2In_ISR:
0118 +1 211 EP2Out_ISR:
0118 +1 212 EP3In_ISR:
0118 +1 213 EP3Out_ISR:
0118 +1 214 EP4In_ISR:
0118 +1 215 EP4Out_ISR:
0118 +1 216 EP5In_ISR:
0118 +1 217 EP5Out_ISR:
0118 +1 218 EP6In_ISR:
0118 +1 219 EP6Out_ISR:
0118 +1 220 EP7In_ISR :
0118 +1 221 EP7Out_ISR:
0118 +1 222 Not_Used: ; Should not get any of these
0118 32 +1 223 RETI
+1 224
0119 +1 225 ClearINT2: ; Tell the hardware that we're done
0119 E591 +1 226 MOV A, EXIF
011B C2E4 +1 227 CLR ACC.4 ; Clear the Interrupt 2 bit
011D F591 +1 228 MOV EXIF, A
011F 22 +1 229 RET
+1 230
0120 +1 231 USBReset_ISR: ; Bus has been Reset, move to DEFAULT state
0120 C0E0 +1 232 PUSH ACC
0122 C200 +1 233 CLR Configured
0124 3119 +1 234 CALL ClearINT2
+1 235 ; No need to clear source of interrupt
0126 D0E0 +1 236 POP ACC
0128 32 +1 237 RETI
+1 238
0129 +1 239 Suspend_ISR: ; SIE detected an Idle bus
0129 C0E0 +1 240 PUSH ACC
012B E587 +1 241 MOV A, PCON
012D 4401 +1 242 ORL A, #1
012F F587 +1 243 MOV PCON, A ; Go to sleep!
0131 00 +1 244 NOP
0132 00 +1 245 NOP ; Wake up here due to a USBResume
0133 00 +1 246 NOP
0134 3119 +1 247 CALL ClearINT2
0136 D0E0 +1 248 POP ACC
0138 32 +1 249 RETI
+1 250
0139 +1 251 WakeUp_ISR: ; Not using external WAKEUP in these examples
+1 252 ; So this must be due to a USBResume
0139 C2DC +1 253 CLR EICON.4 ; Clear the wakeup interrupt source
013B 32 +1 254 RETI
+1 255
```

```
013C      +1 256      SUDAV_ISR:                ; A Setup packet has been received
013C COD0  +1 257      PUSH      PSW                ; Save Registers before the service routine
013E C0E0  +1 258      PUSH      ACC
0140 C082  +1 259      PUSH      DPL
0142 C083  +1 260      PUSH      DPH
0144 3167  +1 261      CALL      ServiceSetupPacket
0146 3119  +1 262      CALL      ClearINT2
                                ; Clear the source of the interrupt
0148 7401  +1 264      MOV       A, #00000001b
014A 907FAB +1 265      ExitISR:MOV    DPTR, #USBIRQ
014D F0     +1 266      MOVX     @DPTR, A
014E D083  +1 267      POP      DPH                ; Restore Registers
0150 D082  +1 268      POP      DPL
0152 D0E0  +1 269      POP      ACC
0154 D0D0  +1 270      POP      PSW
0156 32     +1 271      RETI
                                +1 272
0157      +1 273      SOF_ISR:                ; A Start-Of-Frame packet has been received
0157 COD0  +1 274      PUSH      PSW                ; Save Registers before the service routine
0159 C0E0  +1 275      PUSH      ACC
015B C082  +1 276      PUSH      DPL
015D C083  +1 277      PUSH      DPH
015F 71D7  +1 278      CALL      ServiceTimerRoutine
0161 3119  +1 279      CALL      ClearINT2
                                ; Clear the source of the interrupt
0163 7402  +1 281      MOV       A, #00000010b
0165 80E3  +1 282      JMP      ExitISR
                                +1 283
                                +1 284
                                +1 285
                                +1 286      ;$include (USB_INT.A51)
                                +1 287      ; This module is common to all of the examples.
                                +1 288      ; It services USB Requests from the SIE.
                                +1 289      ; Interpretation of the Output Reports is handled by MAIN
                                +1 290      ;
                                +1 291      CSEG
0167      +1 292      ServiceSetupPacket:
0167 907FE8 +1 293      MOV      DPTR, #SETUPDAT        ; Point to Setup Packet data
016A E0     +1 294      MOVX     A, @DPTR                ; Get the RequestType
016B A2E7  +1 295      MOV      C, ACC.7              ; Bit 7 = 1 means IO device needs to send data
to P
                                C Host
016D 9202  +1 296      MOV      SendData, C
016F 545C  +1 297      ANL     A, #01011100b        ; IF RequestType[6.4.3.2] = 1 THEN goto
BadRequest
0171 7050  +1 298      JNZ     BadRequest
0173 E0     +1 299      MOVX     A, @DPTR                ; IF RequestType[1&0] = 1 THEN goto BadRequest
0174 A2E0  +1 300      MOV      C, ACC.0
0176 82E1  +1 301      ANL     C, ACC.1
0178 4049  +1 302      JC      BadRequest
017A 30E502 +1 303      JNB     ACC.5, NotB5          ; IF RequestType[5] = 1 THEN RequestType[1,0] =
[1,
                                1]
017D 7403  +1 304      MOV      A, #00000011b
017F 5403  +1 305      NotB5: ANL     A, #00000011b        ; Set CommandIndex[5,4] = RequestType[1,0]
0181 C4     +1 306      SWAP    A
0182 F540  +1 307      MOV      Temp, A                ; Save HI nibble of CommandIndex
                                +1 308      ; Set CommandIndex[3,0] = Request[3,0]
0184 A3     +1 309      INC     DPTR                    ; Point to Request
0185 E0     +1 310      MOVX     A, @DPTR
0186 540F  +1 311      ANL     A, #00001111b        ; Only 13 are defined today, handle in table
0188 4540  +1 312      ORL     A, Temp
018A 31D2  +1 313      CALL    CorrectSubroutine      ; goto CommandTable(CommandIndex)
                                +1 314      ; Returns STALL=1 if a stall is required
018C 200134 +1 315      JB      STALL, BadRequest
018F 300218 +1 316      JNB     SendData, HandShake
0192 200320 +1 317      JB      IsDescriptor, LoadSUDPTR; EZ-USB has a short cut for descriptors
                                +1 318      ; Send data in ReplyBuffer
0195 907F02 +1 319      MOV      DPTR, #EP0InBuffer+2
```

```
0198 7846      +1 320          MOV     R0, #ReplyBuffer+3
019A 754003    +1 321          MOV     Temp, #3                ; Copy maximum byte count
019D E6        +1 322      CopyRB: MOV     A, @R0
019E F0        +1 323          MOVX    @DPTR, A
019F 1582      +1 324          DEC     DPL
01A1 18        +1 325          DEC     R0
01A2 D540F8    +1 326          DJNZ   Temp, CopyRB
01A5 E6        +1 327          MOV     A, @R0                ; Get real byte count
01A6          +1 328      SendEP0InBuffer:
01A6 907FB5    +1 329          MOV     DPTR, #In0ByteCount
01A9          +1 330      StartXfer:
01A9 F0        +1 331          MOVX    @DPTR, A                ; This write initiates the transfer
01AA          +1 332      HandShake:                ; Handshake with host
01AA 754002    +1 333          MOV     Temp, #00000010b      ; Set HSNACK to tell the SIE that we're done
01AD          +1 334      SetEP0Control:
01AD 907FB4    +1 335          MOV     DPTR, #EP0Control
01B0 E0        +1 336          MOVX    A, @DPTR
01B1 4540      +1 337          ORL     A, Temp
01B3 F0        +1 338          MOVX    @DPTR, A
01B4 22        +1 339          RET
01B5          +1 340      LoadSUDPTR:                ; Send the data pointed to by DPTR
01B5 858240    +1 341          MOV     Temp, DPL
01B8 E583      +1 342          MOV     A, DPH
01BA 907FD4    +1 343          MOV     DPTR, #SUDPTR
01BD F0        +1 344          MOVX    @DPTR, A
01BE E540      +1 345          MOV     A, Temp
01C0 A3        +1 346          INC     DPTR
01C1 80E6      +1 347          JMP     StartXfer
01C3          +1 348      BadRequest:                ; Invalid Request was received
01C3 754003    +1 349          MOV     Temp, #00000011b      ; Set EPOSTALL and HSNACK
01C6 80E5      +1 350          JMP     SetEP0Control
01C8          +1 351
01C8 E0        +1 352      NextDPTR:                ; Returns (DPTR + byte DPTR is pointing to)
01C8          +1 353          MOVX    A, @DPTR
01C9          +1 354      BumpDPTR:                ; Returns (DPTR + ACC)
01C9 2582      +1 355          ADD     A, DPL
01CB F582      +1 356          MOV     DPL, A
01CD 5002      +1 357          JNC     Skip
01CF 0583      +1 358          INC     DPH                ; Need 16 bit arithmetic here
01D1 22        +1 359      Skip:  RET
01D2          +1 360
01D2          +1 361      CorrectSubroutine:        ; Jump to the subroutine that DPTR is pointing
to
01D2 9001F7    +1 362          MOV     DPTR, #CommandTable
01D5 31C9      +1 363          CALL   BumpDPTR                ; Point to entry
01D7 E0        +1 364          MOVX    A, @DPTR                ; Get the offset
01D8 9001F7    +1 365          MOV     DPTR, #CommandTable
01DB 31C9      +1 366          CALL   BumpDPTR                ; Get the routine address
01DD C082      +1 367          PUSH   DPL                ; Create a RETURN address on stack
01DF C083      +1 368          PUSH   DPH                ; Note: JMP @A+DPTR not used since A, DPTR
needed
01E1 7845      +1 369          MOV     R0, #ReplyBuffer+2
01E3 E4        +1 370          CLR     A
01E4 F6        +1 371          MOV     @R0, A                ; Clear ReplyBuffer
01E5 18        +1 372          DEC     R0
01E6 F6        +1 373          MOV     @R0, A
01E7 18        +1 374          DEC     R0
01E8 7601      +1 375          MOV     @R0, #1                ; Default non-descriptor reply
01EA 907FEA    +1 376          MOV     DPTR, #SETUPDAT+2      ; Point to LOW(wValue)
01ED E0        +1 377          MOVX    A, @DPTR                ; Many of the routines need these
01EE F5F0      +1 378          MOV     B, A                ; LOW(wValue) in B
01F0 A3        +1 379          INC     DPTR
01F1 E0        +1 380          MOVX    A, @DPTR                ; HIGH(wValue) in A
01F2 C201      +1 381          CLR     STALL
01F4 C203      +1 382          CLR     IsDescriptor
01F6 22        +1 383          RET                            ; Go to service routine
01F6          +1 384
01F6          +1 385      ; Since the table only contains byte offsets, it is important that all these routines
are
```

```
+1 386 ; within one page (100H) of CommandTable
+1 387 ;
01F7 +1 388 CommandTable:
+1 389 ; First 16 commands are for the Device
01F7 6C +1 390 DB Device_Get_Status - CommandTable
01F8 40 +1 391 DB Device_Clear_Feature - CommandTable
01F9 40 +1 392 DB Invalid - CommandTable
01FA 40 +1 393 DB Device_Set_Feature - CommandTable
01FB 40 +1 394 DB Invalid - CommandTable
01FC 40 +1 395 DB Invalid - CommandTable ; SIE implements Device_Set_Address
01FD 80 +1 396 DB Get_Descriptor - CommandTable
01FE 40 +1 397 DB Set_Descriptor - CommandTable
01FF 69 +1 398 DB Get_Configuration - CommandTable
0200 73 +1 399 DB Set_Configuration - CommandTable
0201 40 +1 400 DB Invalid - CommandTable
0202 40 +1 401 DB Invalid - CommandTable
0203 40 +1 402 DB Invalid - CommandTable
0204 40 +1 403 DB Invalid - CommandTable
0205 40 +1 404 DB Invalid - CommandTable
0206 40 +1 405 DB Invalid - CommandTable
+1 406 ; Next 16 commands are for the Interface
0207 70 +1 407 DB Interface_Get_Status - CommandTable
0208 40 +1 408 DB Interface_Clear_Feature - CommandTable
0209 40 +1 409 DB Invalid - CommandTable
020A 40 +1 410 DB Interface_Set_Feature - CommandTable
020B 40 +1 411 DB Invalid - CommandTable
020C 40 +1 412 DB Invalid - CommandTable
020D A4 +1 413 DB Get_Class_Descriptor - CommandTable
020E 40 +1 414 DB Set_Class_Descriptor - CommandTable
020F 40 +1 415 DB Invalid - CommandTable
0210 40 +1 416 DB Invalid - CommandTable
0211 40 +1 417 DB Get_Interface - CommandTable
0212 40 +1 418 DB Set_Interface - CommandTable
0213 40 +1 419 DB Invalid - CommandTable
0214 40 +1 420 DB Invalid - CommandTable
0215 40 +1 421 DB Invalid - CommandTable
0216 40 +1 422 DB Invalid - CommandTable
+1 423 ; Next 16 commands are for the Endpoint
0217 70 +1 424 DB Endpoint_Get_Status - CommandTable
0218 42 +1 425 DB Endpoint_Clear_Feature - CommandTable
0219 40 +1 426 DB Invalid - CommandTable
021A 40 +1 427 DB Endpoint_Set_Feature - CommandTable
021B 40 +1 428 DB Invalid - CommandTable
021C 40 +1 429 DB Invalid - CommandTable
021D 40 +1 430 DB Invalid - CommandTable
021E 40 +1 431 DB Invalid - CommandTable
021F 40 +1 432 DB Invalid - CommandTable
0220 40 +1 433 DB Invalid - CommandTable
0221 40 +1 434 DB Invalid - CommandTable
0222 40 +1 435 DB Invalid - CommandTable
0223 40 +1 436 DB Endpoint_Sync_Frame - CommandTable
0224 40 +1 437 DB Invalid - CommandTable
0225 40 +1 438 DB Invalid - CommandTable
0226 40 +1 439 DB Invalid - CommandTable
+1 440 ; Next 16 commands are Class Requests
0227 40 +1 441 DB Invalid - CommandTable
0228 55 +1 442 DB Get_Report - CommandTable
0229 62 +1 443 DB Get_Idle - CommandTable
022A 40 +1 444 DB Get_Protocol - CommandTable
022B 40 +1 445 DB Invalid - CommandTable
022C 40 +1 446 DB Invalid - CommandTable
022D 40 +1 447 DB Invalid - CommandTable
022E 40 +1 448 DB Invalid - CommandTable
022F 40 +1 449 DB Invalid - CommandTable
0230 43 +1 450 DB Set_Report - CommandTable
0231 5C +1 451 DB Set_Idle - CommandTable
```

```
0232 40      +1 452          DB Set_Protocol - CommandTable
0233 40      +1 453          DB Invalid - CommandTable
0234 40      +1 454          DB Invalid - CommandTable
0235 40      +1 455          DB Invalid - CommandTable
0236 40      +1 456          DB Invalid - CommandTable
          +1 457          ;
          +1 458          ; Many requests are INVALID for this example
0237          +1 459          Get_Protocol:          ; We are not a Boot device
0237          +1 460          Set_Protocol:          ; We are not a Boot device
0237          +1 461          Set_Descriptor:        ; Our Descriptors are static
0237          +1 462          Set_Class_Descriptor:   ; Our Descriptors are static
0237          +1 463          Set_Interface:         ; We only have one Interface
0237          +1 464          Get_Interface:         ; We do not have an Alternate setting
0237          +1 465          Device_Set_Feature:    ; We have no features that can be set or cleared
0237          +1 466          Interface_Set_Feature: ; We have no features that can be set or cleared
0237          +1 467          Endpoint_Set_Feature:  ; We have no features that can be set or cleared
0237          +1 468          Device_Clear_Feature:  ; We have no features that can be set or cleared
0237          +1 469          Interface_Clear_Feature; We have no features that can be set or cleared
0237          +1 470          Endpoint_Sync_Frame:   ; We are not an Isonchronous device
          +1 471
0237          +1 472          Invalid:              ; Invalid Request made, STALL the Endpoint
0237 D201    +1 473          SETB     STALL
          +1 474          ;
0239          +1 475          Endpoint_Clear_Feature: ; We have no features that can be set or cleared
          +1 476          ;
0239 22      +1 477          Reply:  RET
          +1 478
023A          +1 479          Set_Report:          ; Host wants to sent us a Report.
          +1 480          ; The ONLY case in this example where host sends data to us
023A 3000FA  +1 481          JNB     Configured, Invalid ; Need to be Configured to do this command
023D 907FC5  +1 482          MOV     DPTR, #Out0ByteCount ; Enable EP0OutBuffer to receive data
0240 F0      +1 483          MOVX    @DPTR, A          ; Any value will do
0241 907FAA  +1 484          MOV     DPTR, #OUT07IRQ    ; Wait for valid data in EP0OutBuffer
0244 E0      +1 485          Wait4D: MOVX    A, @DPTR
0245 5401    +1 486          ANL     A, #00000001b
0247 60FB    +1 487          JZ      Wait4D
0249 F0      +1 488          MOVX    @DPTR, A          ; Clear the interrupt
024A 61AE    +1 489          JMP     ProcessOutputReport ; RETurn via this subroutine
024C          +1 490          Get_Report:          ; Host wants a Report
024C 3000E8  +1 491          JNB     Configured, Invalid ; Need to be Configured to do this command
024F 08      +1 492          INC     R0          ; Point to ReplyBuffer(1)
0250 7618    +1 493          MOV     @R0, #18H        ; Reply with a recognizable (arbitrary) value
0252 22      +1 494          RET
0253          +1 495          Set_Idle:          ; Host wants to tell us how often we should
talk
0253 3000E1  +1 496          JNB     Configured, Invalid ; Need to be Configured to do this command
0256 F541    +1 497          MOV     Idle_Time, A
0258 22      +1 498          RET          ; Handshake with host
0259          +1 499          Get_Idle:          ; Host must have forgotten what he told us to
do
0259 3000DB  +1 500          JNB     Configured, Invalid ; Need to be Configured to do this command
025C 08      +1 501          INC     R0          ; Point to ReplyBuffer(1)
025D A641    +1 502          MOV     @R0, Idle_Time
025F 22      +1 503          RET
0260          +1 504          Get_Configuration:    ; Need to return 0 or 1
0260 300004  +1 505          JNB     Configured, Configuration0
0263          +1 506          Configuration1:      ; Same bit pattern as Device_Get_Status
0263          +1 507          Device_Get_Status:   ; Only two bits of Device Status are defined
0263 08      +1 508          INC     R0          ; Point to ReplyBuffer(1)
0264 7601    +1 509          MOV     @R0, #1          ; Bit 1=Remote Wakeup(=0), Bit 0=Self
Powered(=1)
0266 22      +1 510          RET
0267          +1 511          Configuration0:     ; Same bit pattern as Interface_Get_Status
0267          +1 512          Interface_Get_Status: ; Interface Status is currently defined as 0
0267          +1 513          Endpoint_Get_Status:
0267 7602    +1 514          MOV     @R0, #2
0269 22      +1 515          RET
026A          +1 516          Set_Configuration:    ; Valid values are 0 and 1
026A E5F0    +1 517          MOV     A, B          ; Get LOW(wValue)
```

```
026C 6006      +1  518          JZ      Deconfigured
026E 14        +1  519          DEC     A
026F 70C6      +1  520          JNZ     Invalid
0271 D200      +1  521          SETB   Configured
0273 22        +1  522          RET
0274          +1  523      Deconfigured:
0274 C200      +1  524          CLR     Configured
0276 22        +1  525          RET
0277          +1  526      Get_Descriptor:                ; Host wants to know who/what we are
0277 D203      +1  527          SETB   IsDescriptor
0279 14        +1  528          DEC     A                ; Valid Values are 1, 2 and 3
027A 9002C1    +1  529          MOV     DPTR, #DeviceDescriptor
027D 60BA      +1  530          JZ      Reply
027F 14        +1  531          DEC     A
0280 9002D3    +1  532          MOV     DPTR, #ConfigurationDescriptor
0283 60B4      +1  533          JZ      Reply
0285 14        +1  534          DEC     A
0286 70AF      +1  535          JNZ     Invalid
              +1  536      ; Request is for a String Descriptor
0288 900314    +1  537          MOV     DPTR, #String0        ; Point to String 0
028B E5F0      +1  538          MOV     A, B                ; Get String Index
028D          +1  539      NextString:
028D 601E      +1  540          JZ      FixUpthenReply
028F F540      +1  541          MOV     Temp, A            ; Save String Index
0291 31C8      +1  542          CALL   NextDPTR
0293 E0        +1  543          MOVX   A, @DPTR            ; Get the String Length (= 0 means we're at
Backsto
              p)
0294 60A1      +1  544          JZ      Invalid            ; Asked for a string I don't have
0296 E540      +1  545          MOV     A, Temp
0298 14        +1  546          DEC     A
0299 80F2      +1  547          JMP     NextString        ; Check if we are there yet
029B          +1  548      Get_Class_Descriptor:        ; Valid values are 21H, 22H, 23H for Class
Request
029B D203      +1  549          SETB   IsDescriptor
029D C3        +1  550          CLR     C
029E 9421      +1  551          SUBB   A, #21H
02A0 9002E5    +1  552          MOV     DPTR, #HIDDescriptor
02A3 6094      +1  553          JZ      Reply
02A5 14        +1  554          DEC     A
02A6 9002F5    +1  555          MOV     DPTR, #ReportDescriptor
02A9 608E      +1  556          JZ      Reply
              +1  557      ; DEC     A                ; This example does not use Physical
Descriptors
              +1  558      ; JZ      Send_Physical_Descriptor
02AB 808A      +1  559          JMP     Invalid
              +1  560      ;
              00B6      +1  561      ; Error check: this MUST be on within a page of CommandTable
              +1  562      WithinSamePage EQU $ - CommandTable
              +1  563      ;
02AD          +1  564      FixUpthenReply:                ; EZ-USB Rev D has a String Descriptor bug
              +1  565      ; Need to fill the IN0BUF (@ 7F00H) myself
02AD E0        +1  566          MOVX   A, @DPTR            ; Get the string length
02AE FF        +1  567          MOV     R7, A                ; Save counter
02AF F5F0      +1  568          MOV     B, A
02B1 7800      +1  569          MOV     R0, #LOW(EP0InBuffer) ; PageReg = 7FH = HIGH(EP0InBuffer)
02B3 F2        +1  570      CopySD: MOVX   @R0, A
02B4 08        +1  571          INC     R0
02B5 A3        +1  572          INC     DPTR
02B6 E0        +1  573          MOVX   A, @DPTR
02B7 DFFA      +1  574          DJNZ   R7, CopySD
              +1  575      ; Fixup complete, get back to the program flow
02B9 D0E0      +1  576          POP     ACC                ; Get rid of the return address
02BB D0E0      +1  577          POP     ACC
02BD E5F0      +1  578          MOV     A, B                ; Retrieve byte count
02BF 21A6      +1  579          JMP     SendEP0InBuffer
              580
              581      ;$include (DTables.A51)
              +1  582      ; This module declares the descriptors
```

```
+1 583 ;
+1 584 ; This example has one Device Descriptor with:
+1 585 ;     One Configuration - single IN port and single OUT port
+1 586 ;     One Interface - there is only one method of accessing the ports
+1 587 ;     One HID Descriptor - to make PC host software simpler
+1 588 ;     One Endpoint Descriptor - for HID Input Reports
+1 589 ;     One Report Descriptor - one byte IN and one byte OUT reports
+1 590 ;     Multiple Sting Descriptors - to aid the user
+1 591 ;
----+1 592         CSEG
02C1+1 593 DeviceDescriptor:
02C1 1201+1 594         DB      18, 1           ; Length, Type
02C3 0101+1 595         DW      101H           ; USB Rev 1.1
02C5 000000+1 596         DB      0, 0, 0       ; Class, Subclass and Protocol
02C8 40+1 597         DB      64           ; EP0 size
02C9 4242+1 598         DW      4242H, 1, 1   ; Vendor ID, Product ID and Version
02CB 0001
02CD 0001
02CF 010200+1 599         DB      1, 2, 0       ; Manufacturer, Product & Serial# Names
02D2 01+1 600         DB      1           ; #Configs
02D3+1 601 ConfigurationDescriptor:
02D3 0902+1 602         DB      9, 2           ; Length, Type
02D5 2200+1 603         DB      LOW(ConfigLength), HIGH(ConfigLength)
02D7 010100+1 604         DB      1, 1, 0       ; #Interfaces, Configuration#, Config. Name
02DA 80+1 605         DB      10000000b     ; Attributes = Bus Powered
02DB 32+1 606         DB      50           ; Max. Power is 50x2 = 100mA
02DC+1 607 InterfaceDescriptor:
02DC 0904+1 608         DB      9, 4           ; Length, Type
02DE 000001+1 609         DB      0, 0, 1       ; No alternate setting, HID uses EP1
02E1 03+1 610         DB      3           ; Class = Human Interface Device
02E2 0000+1 611         DB      0, 0       ; Subclass and Protocol
02E4 00+1 612         DB      0           ; Interface Name
02E5+1 613 HIDDescriptor:
02E5 0921+1 614         DB      9, 21H      ; Length, Type
02E7 0001+1 615         DB      0, 1           ; HID Class Specification compliance
02E9 00+1 616         DB      0           ; Country localization (=none)
02EA 01+1 617         DB      1           ; Number of descriptors to follow
02EB 22+1 618         DB      22H          ; And it's a Report descriptor
02EC 1F00+1 619         DB      LOW(ReportLength), HIGH(ReportLength)
02EE+1 620 EndpointDescriptor:
02EE 0705+1 621         DB      7, 5           ; Length, Type
02F0 81+1 622         DB      10000001b     ; Address = IN 1
02F1 03+1 623         DB      00000011b    ; Interrupt
02F2 4000+1 624         DB      64, 0       ; Maximum packet size (this example only uses 1)
02F4 64+1 625         DB      100         ; Poll every 0.1 seconds
      0022+1 626 ConfigLength EQU $ - ConfigurationDescriptor
+1 627
02F5+1 628 ReportDescriptor: ; Generated with HID Tool, copied to here
02F5 0600FF+1 629         DB      6, 0, 0FFH      ; Usage_Page (Vendor Defined)
02F8 0901+1 630         DB      9, 1           ; Usage (I/O Device)
02FA A101+1 631         DB      0A1H, 1       ; Collection (Application)
02FC 1901+1 632         DB      19H, 1       ; Usage_Minimum
02FE 2902+1 633         DB      29H, 2       ; Usage_Maximum
0300 1500+1 634         DB      15H, 0       ; Logical_Minimum (0)
0302 26FF00+1 635         DB      26H, 255, 0 ; Logical_Maximum (255)
0305 7508+1 636         DB      75H, 8       ; Report_Size (8)
0307 9501+1 637         DB      95H, 1       ; Report_Count (1) = Read Address
0309 8102+1 638         DB      81H, 2       ; Input (Data,Var,Abs)
030B 1901+1 639         DB      19H, 1       ; Usage_Minimum
030D 2902+1 640         DB      29H, 2       ; Usage_Maximum
030F 9502+1 641         DB      95H, 2       ; Report_Count (2) = Write Address + Data
0311 9102+1 642         DB      91H, 2       ; Output (Data,Var,Abs)
0313 C0+1 643         DB      0C0H          ; End_Collection
      001F+1 644 ReportLength EQU $-ReportDescriptor
+1 645
0314+1 646 String0: ; Declare the UNICODE strings
```

```

0314 04030904 +1 647          DB      4, 3, 9, 4      ; Only English language strings supported
0318          +1 648      String1:          ; Manufacturer
0318 2C03      +1 649          DB      (String2-String1),3 ; Length, Type
031A 55005300 +1 650          DB      "U",0,"S",0,"B",0," ",0,"D",0,"e",0,"s",0,"i",0,"g",0,"n",0," ",0
031E 42002000
0322 44006500
0326 73006900
032A 67006E00
032E 2000
0330 42007900 +1 651          DB      "B",0,"y",0," ",0,"E",0,"x",0,"a",0,"m",0,"p",0,"l",0,"e",0
0334 20004500
0338 78006100
033C 6D007000
0340 6C006500
0344          +1 652      String2:          ; Product Name
0344 1603      +1 653          DB      (EndOfDescriptors-String2),3
0346 49003200 +1 654          DB      "I",0,"2",0,"C",0," ",0
034A 43002000
034E 42007200 +1 655          DB      "B",0,"r",0,"i",0,"d",0,"g",0,"e",0
0352 69006400
0356 67006500
035A          +1 656      EndOfDescriptors:
035A 0000      +1 657          DW      0          ; Backstop for String Descriptors
          +1 658
          +1 659
          +1 660
          +1 661
          +1 662      ;$include (Main.A51)
          +1 663      ; This module initializes the microcontroller then executes MAIN forever
          +1 664      ;
          +1 665
035C          +1 666      Reset:
035C 7581EB   +1 667          MOV      SP, #235          ; Initialize the Stack at top of internal
memory
035F 75927F   +1 668          MOV      PageReg, #7FH      ; Needed to use MOVX @Ri
          +1 669
0362 78D6     +1 670          MOV      R0, #LOW(USBControl) ; Simulate a disconnect
0364 E2       +1 671          MOVX     A, @R0
0365 54F3     +1 672          ANL      A, #11110011b    ; Clear DISCON, DISCOE
0367 F2       +1 673          MOVX     @R0, A
0368 718C     +1 674          CALL     Wait100msec        ; Give the host time to react
036A E2       +1 675          MOVX     A, @R0          ; Reconnect with this new identity
036B 4406     +1 676          ORL      A, #00000110b    ; Set DISCOE to enable pullup resistor
036D F2       +1 677          MOVX     @R0, A          ; Set RENUM so that 8051 handles USB requests
036E E4       +1 678          CLR      A
036F F520     +1 679          MOV      FLAGS, A        ; Start in Default state
0371          +1 680      InitializeIOSystem: ; This example uses only I2C signals
          +1 681          ; No initialization is required
          +1 682          ; First initialize the USB level
0371          +1 682      InitializeInterruptSystem:
0371 78AC     +1 683          MOV      R0, #LOW(IN07IEN)
0373 F2       +1 684          MOVX     @R0, A          ; Disable interrupts from IN Endpoints 0-7
0374 08       +1 685          INC      R0
0375 F2       +1 686          MOVX     @R0, A          ; Disable interrupts from OUT Endpoints 0-7
0376 08       +1 687          INC      R0
0377 7403     +1 688          MOV      A, #00000011b
0379 F2       +1 689          MOVX     @R0, A          ; Enable (Resume, Suspend,) SOF and SUDAV INTs
037A 08       +1 690          INC      R0
037B 7401     +1 691          MOV      A, #00000001b
037D F2       +1 692          MOVX     @R0, A          ; Enable Auto Vectoring for USB interrupts
037E 78AA     +1 693          MOV      R0, #LOW(OUT07IRQ)
0380 74FF     +1 694          MOV      A, #0FFH
0382 F2       +1 695          MOVX     @R0, A          ; Clear out any pending interrupts
          +1 696          ; Now enable the main level
0383 75E801   +1 697          MOV      EIE, #00000001b ; Enable INT2 = USB Interrupt (only)
0386 75A8C0   +1 698          MOV      EI, #11000000b   ; Enable interrupt subsystem (and Ser1 for
Dscope)
          +1 699
          +1 700      ; Initialization Complete.

```

```
+1 701 ;
0389 +1 702 MAIN:
0389 00 +1 703 NOP ; Not much of a main loop for this example
038A 80FD +1 704 JMP MAIN ; All actions are initiated by interrupts
+1 705 ; We are a slave, we wait to be told what to do
+1 706
038C +1 707 Wait100msec:
038C 754064 +1 708 MOV Temp, #100
038F +1 709 Wait1msec: ; A delay loop
038F 90FB50 +1 710 MOV DPTR, #-1200
0392 A3 +1 711 More: INC DPTR ; 3 cycles
0393 E582 +1 712 MOV A, DPL ; + 2
0395 4583 +1 713 ORL A, DPH ; + 2
0397 70F9 +1 714 JNZ More ; + 3 = 10 cycles x 1200 = 1msec
0399 D540F3 +1 715 DJNZ Temp, Wait1msec
039C 22 +1 716 RET
+1 717
039D +1 718 Wait4Stop: ; Make sure the STOP from a previous
039D E2 +1 719 MOVX A, @R0 ; I2C transaction has completed
039E 20E6FC +1 720 JB ACC.6, Wait4Stop
03A1 22 +1 721 RET
03A2 +1 722 SendI2Caddress: ; Send the first byte to I2C bus
03A2 719D +1 723 CALL Wait4Stop
03A4 7480 +1 724 MOV A, #10000000b ; Set the START bit
03A6 F2 +1 725 MOVX @R0, A ; I2C engine now primed
03A7 +1 726 SendI2Cdatabyte:
03A7 E0 +1 727 MOVX A, @DPTR ; Get data and . . .
03A8 F3 +1 728 MOVX @R1, A ; . . . send using I2C engine
03A9 +1 729 Wait4Done:
03A9 E2 +1 730 MOVX A, @R0
03AA 30E0FC +1 731 JNB ACC.0, Wait4Done ; Wait for the data to be transmitted
03AD 22 +1 732 RET
+1 733
03AE +1 734 ProcessOutputReport: ; A Report has just been received
+1 735 ; The report is two bytes long
+1 736 ; Byte 1, Bits[7:1] = an I2C Address, Bit[0] = direction (0 = Write)
+1 737 ; Byte 2 = DataByte if byte 1 specifies a write
03AE 78A5 +1 738 MOV R0, #LOW(I2CControl) ; Initialize the pointers to be used
03B0 79A6 +1 739 MOV R1, #LOW(I2CData)
03B2 907EC0 +1 740 MOV DPTR, #EP0OutBuffer ; Point to the Report
03B5 71A2 +1 741 CALL SendI2Caddress
03B7 E0 +1 742 MOVX A, @DPTR ; Retrieve Report Byte 1
03B8 20E007 +1 743 JB ACC.0, I2C_Read
03BB +1 744 I2C_Write:
03BB A3 +1 745 INC DPTR ; Point to data byte to write
03BC 71A7 +1 746 CALL SendI2Cdatabyte
03BE 7440 +1 747 MOV A, #01000000b ; Set the STOP bit
03C0 F2 +1 748 MOVX @R0, A
03C1 22 +1 749 RET
03C2 +1 750 I2C_Read:
03C2 7420 +1 751 MOV A, #00100000b ; Set the LASTRD bit
03C4 F2 +1 752 MOVX @R0, A
03C5 E3 +1 753 MOVX A, @R1 ; Dummy read of I2C_Data
03C6 71A9 +1 754 CALL Wait4Done
03C8 7440 +1 755 MOV A, #01000000b ; Set the STOP bit
03CA F2 +1 756 MOVX @R0, A ; Immediately follow with . . .
03CB E3 +1 757 MOVX A, @R1 ; . . . read the I2C bus data
+1 758 ; Fall into CreateInputReport
03CC +1 759 CreateInputReport:
+1 760 ; The report is only one byte long in this example
+1 761 ; It contains the byte read from the I2C bus
03CC 907E80 +1 762 MOV DPTR, #EP1InBuffer ; Point to the buffer
03CF F0 +1 763 MOVX @DPTR, A ; Update the Report
03D0 907FB7 +1 764 MOV DPTR, #IN1ByteCount
03D3 7401 +1 765 MOV A, #1
03D5 F0 +1 766 MOVX @DPTR, A ; Endpoint 1 now 'armed', next IN will get data
```

```
03D6 22      +1  767          RET
              +1  768
              769
              770      ;$include (Timer.A51)
+1  771      ; This module services the real time interrupt
+1  772      ;
+1  773      ; Get a Real Time interrupt every One millisecond (using SOF interrupt)
+1  774      ;
+1  775      ; HID devices work on a 4 millisecond timer
+1  776      ;
+1  777      ; There is nothing to do in this example!
03D7          +1  778      ServiceTimerRoutine:
03D7 22      +1  779          Ret
              +1  780
              781
              782
              783      END
```