

MACRO ASSEMBLER A51 V6.10
 OBJECT MODULE PLACED IN .\D12.OBJ
 ASSEMBLER INVOKED BY: C:\Keil\C51\BIN\A51.EXE .\D12.A51 REGISTERBANK(0) SET(SMALL) DEBUG EP

```

LOC OBJ          LINE    SOURCE
                1      NAME    InterfaceToD12
                2      ; Created from BAL Version 3.0
                3      ;
                4      ; Copyright (C) 2001, Intel Corporation
                5      ; All rights reserved.
                6      ; Permission is hereby granted to merge this program code with other program
                7      ; material to create a derivative work. This derivative work may be distributed
                8      ; in compiled object form only. Any other publication of this program, in any form,
                9      ; without the explicit permission of the copyright holder is prohibited.
               10      ;
               11      ; Send questions and comments to John.Hyde@intel.com
               12      ;
               13      ;
               14      ; This program uses an attached D12 peripheral for the USB connection and
               15      ; not the integrated SIE - it shows an alternate implementation for those
               16      ; designs that use a microcontroller without an integrated SIE.
               17      ; The D12 is connected to the EZ-USB Address/Data/Control bus so that the
               18      ; ports are free for other applications (therefore the ActiveWire board
               19      ; cannot be used in this example)
    4001          20      D12address    EQU    4001H          ; Command Register
               21      ; The D12 generates an interrupt on INTO
               22      ;
               23      ; This example creates a 1 byte report every 10msec which is a counter.
               24      ; It will accept a 1 byte report that sets this counter.
               25      ; (This test mechanism uses no hardware resources and can easily be
               26      ; expanded to be the basis of all of the other examples)
               27      ;
               28      ;
               29      ; Changes from Version 2.0
               30      ; a) Two misplaced labels corrected
               31      ; b) CommandTable moved outside of the constrained page
               32      ; c) SETUPDAT buffer copied to direct access memory, simplified coding
               33      ; d) Optional Set_Idle now not supported (returns STALL, not ignorred)
               34      ; e) R7 used in place of Temp, saved code space
               35      ; f) Code reorganized to separate hardware dependant sections
               36      ; Old New
               37      ; USBINT.A51 Decode.A51
               38      ; Vectors.A51 ISRs.A51
               39      ; Timer.A51 ISRs.A51
               40      ; Main.A51 Main.A51
               41      ; g) EP0Size made an equate to ease coding of other components
    0010          42      EP0Size    EQU    16          ; For D12
               43      ; h) Code added for descriptors > EP0Size
               44      ;
               45      ; Changes from Version 1.0
               46      ; a) Register saving removed from Vectors.A51
               47      ; Main has no context which needs to be saved
               48      ; b) There was a race condition in USB_INT::Set_Report: which
               49      ; could cause OLD data to be read. Busy algorithm changed
               50      ; c) The USB Version# was incorrectly declared in the Device Descriptor
               51      ; It was 0101H and is now 0110H (data from USB IF)
               52      ; d) Three conditional assemble flags added for different hardware
    0001          53      ANCHOR    EQU    1          ; Set one of these flags to 1
    0000          54      USBSIMM  EQU    0          ; ONLY set one, other should be 0
               55      ; These flags change some initialization and run-time IO.
               56      ;
               57      ;$include (Declare.A51)
    +1           58      ; This module declares the variables and constants used in the examples
  
```

```

+1 59 ; It is common to all of the examples
+1 60 ;
+1 61 ; Declare Special Function Registers used
0088 +1 62 TimerControl DATA 088H
0089 +1 63 TimerMode DATA 089H
008C +1 64 Timer0High DATA 08CH
00A8 +1 65 EI DATA 0A8H
00E8 +1 66 EIE DATA 0E8H ; EZ-USB specific
0091 +1 67 EXIF DATA 091H ; EZ-USB specific
00D8 +1 68 EICON DATA 0D8H ; EZ-USB specific
0092 +1 69 PageReg DATA 092H ; EZ-USB specific, used with MOVX @Ri
0086 +1 70 DPS DATA 086H ; EZ-USB specific, used with dual data pointers
+1 71 ;
+1 72 ; "External" memory locations used, EZ-USB specific
+1 73 ; Note that most of these variables are in Page 7FH
7FE8 +1 74 SETUPDAT EQU 07FE8H
7FD4 +1 75 SUDEPTR EQU 07FD4H
7FB4 +1 76 EP0Control EQU 07FB4H
7F00 +1 77 EP0InBuffer EQU 07F00H
7EC0 +1 78 EP0OutBuffer EQU 07EC0H ; Not in Page 7FH
7E80 +1 79 EP1InBuffer EQU 07E80H ; Not in Page 7FH
7FB5 +1 80 IN0ByteCount EQU 07FB5H
7FC5 +1 81 Out0ByteCount EQU 07FC5H
7FB7 +1 82 IN1ByteCount EQU 07FB7H
7FAC +1 83 IN07IEN EQU 07FACH
7FA9 +1 84 IN07IRQ EQU 07FA9H
7FAD +1 85 OUT07IEN EQU 07FADH
7FAA +1 86 OUT07IRQ EQU 07FAAH
7FAE +1 87 USBIEN EQU 07FAEH
7FAB +1 88 USBIRQ EQU 07FABH
7FD6 +1 89 USBControl EQU 07FD6H
7FA6 +1 90 I2CData EQU 07FA6H
7FA5 +1 91 I2CControl EQU 07FA5H
7F93 +1 92 PortA_Config EQU 07F93H
7F94 +1 93 PortB_Config EQU 07F94H
7F95 +1 94 PortC_Config EQU 07F95H
7F96 +1 95 PortA_OUT EQU 07F96H
7F97 +1 96 PortB_OUT EQU 07F97H
7F98 +1 97 PortC_OUT EQU 07F98H
7F99 +1 98 PortA_PINS EQU 07F99H
7F9A +1 99 PortB_PINS EQU 07F9AH
7F9B +1 100 PortC_PINS EQU 07F9BH
7F9C +1 101 PortA_OE EQU 07F9CH
7F9D +1 102 PortB_OE EQU 07F9DH
7F9E +1 103 PortC_OE EQU 07F9EH
+1 104 ;
+1 105 ; Byte Variables
+1 106
---- +1 107 DSEG AT 20H
0020 +1 108 FLAGS: DS 1 ; This register is bit-addressable
+1 109 ; Bit Variables
0000 +1 110 Configured EQU FLAGS.0 ; Is this device configured
0001 +1 111 STALL EQU FLAGS.1 ; Need to STALL endpoint 0
0002 +1 112 SendData EQU FLAGS.2 ; Need to send data to PC Host
0003 +1 113 IsDescriptor EQU FLAGS.3 ; Data is a descriptor
0004 +1 114 SetAddress EQU FLAGS.4 ; Set the SIE address
+1 115 ;
0021 +1 116 MonitorSpace: DS 1FH ; Used by Dscope
0040 +1 117 CurrentConfiguration:
0040 +1 118 DS 1 ; Some examples support > 1 configurations
0041 +1 119 Counter: DS 1 ; A counter for timed Reports
0042 +1 120 SaveDPH: DS 1 ; Need to save Descriptor Pointer ..
0043 +1 121 SaveDPL: DS 1 ; .. for descriptors > 16 bytes
0044 +1 122 SaveLength: DS 1 ; Number of bytes still to send
0045 +1 123 SetupDataResv: DS 1 ; Reserved byte (for D12)
0046 +1 124 SetupDataLen: DS 1 ; Length of Setup Data

```

```

0047      +1 125      SetupData:          ; Buffer in direct access memory
0047      +1 126      RequestType:   DS      1      ; USB Setup Packet
0048      +1 127      Request:       DS      1      ; USB Setup Packet
0049      +1 128      wValueLow:     DS      1      ; USB Setup Packet
004A      +1 129      wValueHigh:    DS      1      ; USB Setup Packet
004B      +1 130      wIndexLow:     DS      1      ; USB Setup Packet
004C      +1 131      wIndexHigh:    DS      1      ; USB Setup Packet
004D      +1 132      wLengthLow:    DS      1      ; USB Setup Packet
004E      +1 133      wLengthHigh:   DS      1      ; USB Setup Packet
004F      +1 134      ReplyCount:    DS      1      ; Byte count of following buffer
0050      +1 135      ReplyBuffer:   DS      16     ; = D12 EP0size
          +1 136
          +1 137      ;
          +1 138      ; Declare the specific variables used by each of the examples
0060      +1 139      Msec_Counter:  DS      1      ; Counts up to 8 msec
          +1 140
          +1 141
          +1 142      ;$include (ISRs.A51)
          +1 143      ; This module contains all the EZUSB-specific hardware code
          +1 144      ; This module also contains all of the interrupt vector declarations and
          +1 145      ; the first level interrupt servicing (register save, call subroutine,
          +1 146      ; clear interrupt source, restore registers, return)
          +1 147      ; Suspend and Resume are handled totally in this module
          +1 148      ;
          +1 149      ; A Reset sends us to Program space location 0
-----  +1 150          CSEG AT 0          ; Code space
          +1 151          USING 0          ; Reset forces Register Bank 0
0000 020221 +1 152          LJMP Reset
          +1 153      ;
          +1 154      ; The interrupt vector table is also located here
          +1 155      ; EZ-USB has two levels of USB interrupts:
          +1 156      ; 1-the main level is described in this table (at ORG 43H)
          +1 157      ; 2-there are 21 sources of USB interrupts and these are described in USB_ISR
          +1 158      ; This means that two levels of acknowledgement and clearing will be required
0003 0200F7 +1 159          LJMP INT0_ISR          ; Features not used are commented out
          +1 160      ; ORG 0BH
          +1 161      ; LJMP Timer0_ISR
          +1 162      ; ORG 13H
          +1 163      ; LJMP INT1_ISR
          +1 164      ; ORG 1BH
          +1 165      ; LJMP Timer1_ISR
          +1 166      ; ORG 23H
          +1 167      ; LJMP UART0_ISR
          +1 168      ; ORG 2BH
          +1 169      ; LJMP Timer2_ISR
          +1 170      ; ORG 33H
          +1 171      ; LJMP WakeUp_ISR
          +1 172      ; ORG 3BH
          +1 173      ; LJMP UART1_ISR
          +1 174      ; ORG 43H
          +1 175      ; LJMP USB_ISR          ; Auto Vector will replace byte 45H
          +1 176      ; ORG 4BH
          +1 177      ; LJMP I2C_ISR
          +1 178      ; ORG 53H
          +1 179      ; LJMP INT4_ISR
          +1 180      ; ORG 5BH
          +1 181      ; LJMP INT5_ISR
          +1 182      ; ORG 63H
          +1 183      ; LJMP INT6_ISR
          +1 184
00E0      +1 185          ORG 0E0H          ; Keep out of the way of dScope monitor
          +1 186      ; If not using dScope then this memory hole may be used for useful routines.
          +1 187      ; End of Interrupt Vector tables
          +1 188
          +1 189      ; When a feature is used insert the required interrupt processing here
          +1 190      ; The example use only used Endpoints 0 and 1 and also SOF for timing

```

```

00E0      +1 191      Reserved:
00E0      +1 192      Timer0_ISR:
00E0      +1 193      INT1_ISR:
00E0      +1 194      Timer1_ISR:
00E0      +1 195      UART0_ISR:
00E0      +1 196      Timer2_ISR:
00E0      +1 197      UART1_ISR:
00E0      +1 198      I2C_ISR:
00E0      +1 199      INT4_ISR:
00E0      +1 200      INT5_ISR:
00E0      +1 201      INT6_ISR:
00E0      +1 202      Not_Used:                ; Should not get any of these
00E0 32    +1 203      RETI
           +1 204
           +1 205      ; Declare the support routines that talk to the D12 component
00E1      +1 206      D12write:
00E1 F0    +1 207      MOVX      @DPTR, A
00E2 00    +1 208      NOP                ; Delay a little so that we don't violate
00E3 00    +1 209      NOP                ; the back-to-back timing of D12
00E4 00    +1 210      NOP
00E5 00    +1 211      NOP
00E6 22    +1 212      RET
00E7      +1 213      D12command:          ; Send command to D12
00E7 904001 +1 214      MOV      DPTR, #D12address ; Point to D12 Command Register
00EA 11E1  +1 215      CALL     D12write
00EC 1582  +1 216      DEC      DPL                ; Return with DPTR -> D12 Data Register
00EE 22    +1 217      RET
00EF      +1 218      D12GetStatus:        ; Get Status. Also clears interrupt
00EF 11E7  +1 219      CALL     D12command
           +1 220      ; Fall into D12read
00F1      +1 221      D12read:          ; Input from D12 component
00F1 E0    +1 222      MOVX      A, @DPTR
00F2 00    +1 223      NOP                ; Delay a little so that we don't violate
00F3 00    +1 224      NOP                ; the back-to-back timing of D12
00F4 00    +1 225      NOP
00F5 00    +1 226      NOP
00F6 22    +1 227      RET
           +1 228
00F7      +1 229      INTO_ISR:          ; D12 has just interrupted us:
00F7 74F4  +1 230      MOV      A, #0F4H        ; Read_Interrupt_Register command
00F9 11EF  +1 231      CALL     D12GetStatus
00FB 11FE  +1 232      CALL     FindInterruptSource
           +1 233      ; If INTO is still LOW we will get interrupted again
00FD 32    +1 234      RETI
           +1 235
00FE      +1 236      FindInterruptSource:    ; Check each possible interrupt source
           +1 237      ; Check in order of importance
00FE 20E735 +1 238      JB      ACC.7, SuspendChange
0101 20E630 +1 239      JB      ACC.6, BusReset
0104 20E050 +1 240      JB      ACC.0, EP0Out
0107 20E13D +1 241      JB      ACC.1, EP0In
010A 20E22A +1 242      JB      ACC.2, EP1Out
010D 20E333 +1 243      JB      ACC.3, EP1In
0110 20E428 +1 244      JB      ACC.4, EP2Out
0113 20E529 +1 245      JB      ACC.5, EP2In
           +1 246      ; An interrupt with no bits set is SOF
0116      +1 247      SOF_ISR:          ; A Start-Of-Frame packet has been received
           +1 248      ; This routine services the real time interrupt
           +1 249      ;
0116      +1 250      ServiceTimerRoutine:
           +1 251      JNB     Configured, Done        ; Need to be Configured to respond
0119 D5601A +1 252      DJNZ    Msec_counter, Done        ; Only need to update every 8msec
011C 756008 +1 253      MOV     Msec_counter, #8        ; Reinitialize
           +1 254      ;
           +1 255      ; Create an Input Report from the counter value
           +1 256      ; This will be continually overwritten while the PCHost is not polling for data

```

```
011F 0541      +1 257          INC      Counter
0121          +1 258      CreateInputReport:
                +1 259          ; The report is only one byte long in this example
                +1 260          ; It contains a new value for the counter
0121 7443      +1 261          MOV      A, #43H          ; Set_Status(EP1In)
0123 11E7      +1 262          CALL     D12Command
0125 E4        +1 263          CLR      A          ; Clear EP1In so we can overwrite it
0126 11E1      +1 264          CALL     D12write
0128 7403      +1 265          MOV      A, #03          ; Select_Endpoint(EP1In) command
012A 11E7      +1 266          CALL     D12command
012C 754F01    +1 267          MOV      ReplyCount, #1      ; 1 byte of data
012F 854150    +1 268          MOV      ReplyBuffer, Counter
0132 21D3      +1 269          JMP      SendReplyBuffer      ; RETURN via SendReplyBuffer
                +1 270
0134          +1 271      BusReset:
0134 C200      +1 272          CLR      Configured      ; D12 automatically sets address to 0
0136          +1 273      SuspendChange:
0136 22        +1 274      Done:  RET
                +1 275
0137 7442      +1 276      EP1Out: MOV     A, #42H          ; Read_Last_Transaction_Status(EP1Out)
0139 80B4      +1 277          JMP      D12GetStatus      ; Should not get one of these
013B 7444      +1 278      EP2Out: MOV     A, #44H          ; Read_Last_Transaction_Status(EP2Out)
013D 80B0      +1 279          JMP      D12GetStatus      ; Should not get one of these
013F 7445      +1 280      EP2In:  MOV     A, #45H          ; Read_Last_Transaction_Status(EP2In)
0141 80AC      +1 281          JMP      D12GetStatus      ; Should not get one of these
                +1 282
0143 7443      +1 283      EP1In:  MOV     A, #43H          ; Read_Last_Transaction_Status(EP1In)
0145 80A8      +1 284          JMP      D12GetStatus      ; PC has just read latest Input Report
                +1 285
0147          +1 286      EP0In:
0147 7441      +1 287          MOV     A, #41H          ; Read_Last_Transaction_Status(EP0In)
0149 11EF      +1 288          CALL     D12GetStatus
                +1 289          ; Do I need this buffer to send a pending partial descriptor?
014B E544      +1 290          MOV     A, SaveLength
014D 60E7      +1 291          JZ      Done
                +1 292          ; Yes, I have data that I should prepare for the PC Host
014F 854283    +1 293          MOV     DPH, SaveDPH
0152 854382    +1 294          MOV     DPL, SaveDPL
0155 8052      +1 295          JMP      SendNextPieceOfDescriptor
                +1 296
0157 7440      +1 297      EP0Out: MOV     A, #40H          ; Read_Last_Transaction_Status(EP0Out)
0159 11EF      +1 298          CALL     D12GetStatus
015B          +1 299      SetupPacketRecieved:
015B          +1 300      SUDAV_ISR:          ; A Setup packet has been received
                +1 301          ; EP0Out data packets are collected elsewhere in this program
015B E4        +1 302          CLR      A
015C F544      +1 303          MOV     SaveLength, A      ; Clear any pending transactions (if any)
                +1 304          ; MOV     A, #0          ; Select_Endpoint(EP0Out) command
015E 11E7      +1 305          CALL     D12command
0160 74F0      +1 306          MOV     A, #0F0H          ; Read_Buffer command
0162 11E7      +1 307          CALL     D12command
0164 7845      +1 308          MOV     R0, #SetupDataResv      ; Copy packet to direct access memory
0166 7F0A      +1 309          MOV     R7, #10
0168 11F1      +1 310      CopySD: CALL     D12read
016A F6        +1 311          MOV     @R0, A
016B 08        +1 312          INC     R0
016C DFFA      +1 313          DJNZ   R7, CopySD
                +1 314          ; Setup Packet successfully received. Thank the D12
016E 74F1      +1 315          MOV     A, #0F1H          ; Acknowledge_Setup command
0170 11E7      +1 316          CALL     D12command
0172 74F2      +1 317          MOV     A, #0F2H          ; Clear_Buffer command
0174 11E7      +1 318          CALL     D12command
0176 7401      +1 319          MOV     A, #01          ; Select_Endpoint(EP0In) command
0178 11E7      +1 320          CALL     D12command
017A 74F1      +1 321          MOV     A, #0F1H          ; Acknowledge_Setup command
017C 11E7      +1 322          CALL     D12command
```

```

+1 323 ; The D12 can now receive and send other packets
+1 324 ; Note that EP0In is selected for the response
017E 5169 +1 325 CALL ServiceSetupPacket
+1 326 ; Handle the decode of the Setup packet
+1 327 ; if STALL {Stall the endpoint}
+1 328 ; else if SetAddress {Update SIE address}
+1 329 ; else if SendData {
+1 330 ;     if IsDescriptor {send DPTR->descriptor, A = length}
+1 331 ;     else {send ReplyBuffer}
+1 332 ; }
0180 300110 +1 333 JNB STALL, Continue
0183 +1 334 SendSTALL: ; Invalid Request was received
0183 7440 +1 335 MOV A, #40H ; Set_Status(EP0Out) command
0185 11E7 +1 336 CALL D12command
0187 7401 +1 337 MOV A, #00000001b ; Set STALL bit
0189 11E1 +1 338 CALL D12write
018B 7441 +1 339 MOV A, #41H ; Set_Status(EP0In) command
018D 11E7 +1 340 CALL D12command
018F 7401 +1 341 MOV A, #00000001b ; Set STALL bit
0191 01E1 +1 342 JMP D12write ; RETurn via D12write
0193 +1 343 Continue:
0193 200457 +1 344 JB SetAddress, SetD12Address
0196 30026E +1 345 JNB SendData, HandShake
0199 300337 +1 346 JNB IsDescriptor, SendReplyBuffer
+1 347 ; Copy up to 16 bytes that DPTR is pointing to into ReplyBuffer
019C FF +1 348 MOV R7, A ; Save LENGTH
+1 349 ; Need to return the smaller of "Requested Length" and "Actual Length"
+1 350 ; If "Requested Length" > 255 then use "Actual Length"
+1 351 ; There are no descriptors > 255 in this example
019D E54E +1 352 MOV A, wLengthHigh
019F 7007 +1 353 JNZ UseActual
01A1 C3 +1 354 CLR C
01A2 954D +1 355 SUBB A, wLengthLow
01A4 E54D +1 356 MOV A, wLengthLow ; This does not affect Carry
01A6 5001 +1 357 JNC UsewLengthLow
01A8 +1 358 UseActual:
01A8 EF +1 359 MOV A, R7
01A9 +1 360 UsewLengthLow:
01A9 +1 361 SendNextPieceOfDescriptor: ; DPTR -> Descriptor to be sent
01A9 FF +1 362 MOV R7, A ; Save LENGTH again
01AA 754400 +1 363 MOV R7, #0 ; Default case, overwrite if necessary
+1 364 ; Do I have more than EP0Size bytes to send?
01AD C3 +1 365 CLR C
01AE 9410 +1 366 SUBB A, #EP0Size
01B0 4015 +1 367 JC SendPacketToD12
+1 368 ; Need to split >EP0Size bytes into multiple packets
+1 369 ; Calculate position of next packet and send the first 16 bytes now
01B2 F544 +1 370 MOV SaveLength, A ; Send the other bytes later
01B4 7F10 +1 371 MOV R7, #EP0Size
01B6 C083 +1 372 PUSH DPH ; Save current pointer
01B8 C082 +1 373 PUSH DPL
01BA EF +1 374 MOV A, R7
01BB 51AC +1 375 CALL BumpDPTR ; Calculate next DPTR
01BD 858342 +1 376 MOV SaveDPH, DPH ; Save next pointer
01C0 858243 +1 377 MOV SaveDPL, DPL
01C3 D082 +1 378 POP DPL ; Retrieve current pointer
01C5 D083 +1 379 POP DPH
01C7 +1 380 SendPacketToD12: ; There are no other packets after this one
01C7 EF +1 381 MOV A, R7 ; Retrieve length
01C8 784F +1 382 SP2: MOV R0, #ReplyCount
01CA F6 +1 383 MOV @R0, A ; Set count
01CB 6006 +1 384 JZ SendReplyBuffer ; Special case = zero length packet
01CD 08 +1 385 CopySTD:INC R0 ; Point into ReplyBuffer
01CE E0 +1 386 MOVX A, @DPTR ; Get the descriptor byte to send
01CF F6 +1 387 MOV @R0, A ; Copy it into ReplyBuffer
01D0 A3 +1 388 INC DPTR

```

```
01D1 DFFA      +1 389          DJNZ    R7, CopySTD
01D3          +1 390      SendReplyBuffer:      ; Send data in ReplyBuffer
01D3 74F0      +1 391          MOV     A, #0F0H      ; Write_Buffer command
01D5 11E7      +1 392          CALL   D12command
01D7 E4        +1 393          CLR     A
01D8 11E1      +1 394          CALL   D12write      ; Start with the reserved byte = 0
01DA 784F      +1 395          MOV     R0, #ReplyCount ; Point to byte count
01DC E6        +1 396          MOV     A, @R0
01DD 11E1      +1 397          CALL   D12write      ; Write byte count
01DF 6007      +1 398          JZ     ValidateBuffer ; Special case = zero length packet
01E1 FF        +1 399          MOV     R7, A        ; Save in R7 Loop Counter too
01E2 08        +1 400      CopyRB: INC     R0      ; Point to first data byte
01E3 E6        +1 401          MOV     A, @R0
01E4 11E1      +1 402          CALL   D12write      ; Write data bytes
01E6 DFFA      +1 403          DJNZ   R7, CopyRB
01E8          +1 404      ValidateBuffer:      ; Get this packet on it's way
01E8 74FA      +1 405          MOV     A, #0FAH     ; Validate_Buffer command
01EA 11E7      +1 406          CALL   D12command    ; D12 will respond to the next IN token
01EC 22        +1 407          RET
01ED          +1 408
01ED          +1 409      SetD12Address:
01ED          +1 410          ; Note that the D12 will defer update of it's device address until it
01ED          +1 411          ; sends the handshake packet. Address MUST be written BEFORE handshake
01ED 74D0      +1 412          MOV     A, #0D0H     ; Set_Address command
01EF 11E7      +1 413          CALL   D12command
01F1 E549      +1 414          MOV     A, wValueLow
01F3 D2E7      +1 415          SETB   ACC.7        ; Set address and enable this device
01F5 11E1      +1 416          CALL   D12write
01F7 74D8      +1 417          MOV     A, #0D8H     ; Enable_Endpoints command
01F9 11E7      +1 418          CALL   D12command
01FB 7401      +1 419          MOV     A, #01H     ; Enable other (non-control) endpoints
01FD 11E1      +1 420          CALL   D12write
01FF 74FB      +1 421          ; Need to set the interrupt up such that we can detect a SOF
01FF 74FB      +1 422          MOV     A, #0FBH     ; Set_DMA mode command
0201 11E7      +1 423          CALL   D12command
0203 7420      +1 424          MOV     A, #00100000b ; Enable SOF to generate an interrupt
0205 11E1      +1 425          CALL   D12write
0207          +1 426          ; Fall into HandShake
0207          +1 427      HandShake:          ; Handshake with host
0207 E4        +1 428          CLR     A            ; Send a zero length packet
0208 80BE      +1 429          JMP     SP2          ; SendPacketToD12 (after MOV A, R7)
020A          +1 430
020A          +1 431      ProcessOutputReport: ; A Report has just been received
020A          +1 432          ; This is the only time, in this example, that data is received from the PC Host
020A 74F4      +1 433      Wait4D: MOV     A, #0F4H ; Read_Interrupt_Register command
020C 11EF      +1 434          CALL   D12GetStatus
020E 30E0F9    +1 435          JNB    ACC.0, Wait4D ; Wait for the data packet
0211 E4        +1 436          CLR     A            ; Select_Endpoint(EP0Out) command
0212 11E7      +1 437          CALL   D12command
0214 74F0      +1 438          MOV     A, #0F0H     ; Read_Buffer command
0216 11E7      +1 439          CALL   D12command
0218 11F1      +1 440          CALL   D12read      ; Read dummy byte
0218          +1 441          ; The report is only one byte long in this D12 example
0218          +1 442          ; It contains a new value for the Counter
021A 11F1      +1 443          CALL   D12read      ; Read Count (=1)
021C 11F1      +1 444          CALL   D12read      ; Read data value
021E F541      +1 445          MOV     Counter, A
0220 22        +1 446          RET
0220          +1 447
0220          +1 448
0220          +1 449
0220          +1 450
0220          +1 451          ;$include (Main.A51)
0220          +1 452          ; This module initializes the microcontroller then executes MAIN forever
0220          +1 453          ; It is hardware dependant
0220          +1 454
```

```

0221      +1 455      Reset:
0221 7581DF +1 456      MOV     SP, #0DFH          ; Initialize the Stack
0224 75927F +1 457      MOV     PageReg, #7FH       ; Allows MOVX Ri to access EZ-USB memory
0227 E4     +1 458      CLR     A
0228 F520   +1 459      MOV     FLAGS, A          ; Start in Default state
022A F544   +1 460      MOV     SaveLength, A     ; No partial descriptor to send
022C 04     +1 461      INC     A
022D F541   +1 462      MOV     Counter, A
022F       +1 463      InitializeMsecCounter:
022F F560   +1 464      MOV     Msec_counter, A
0231       +1 465      InitializeIOSystem:          ; Work around the dScope monitor I/O needs
                                ; Set up PC2 as an INTO# input
0231       +1 466      ;
0231 789E   +1 467      MOV     R0, #LOW(PortC_OE)   ; PageReg = 7F = HIGH(PortC_OE)
0233 E2     +1 468      MOVX    A, @R0              ; Get current direction bits
0234 C2E2   +1 469      CLR     ACC.2
0236 F2     +1 470      MOVX    @R0, A          ; Set bit2 as input
0237 7895   +1 471      MOV     R0, #LOW(PortC_Config)
0239 E2     +1 472      MOVX    A, @R0              ; Get current configuration
023A D2E2   +1 473      SETB   ACC.2
023C F2     +1 474      MOVX    @R0, A          ; Set bit2 as alternate function
                                ; No additional initialization required for this D12 test example
023D       +1 475      InitializeD12:
023D 7F64   +1 476      MOV     R7, #100          ; Give D12 time to power up
023F 5159   +1 477      CALL   Wait1msec
0241 74F3   +1 478      MOV     A, #0F3H         ; Set_Mode command
0243 11E7   +1 479      CALL   D12command
                                ; Use easy values during initial program development, these WILL change later
0245 7416   +1 480      MOV     A, #00010110b    ; Endpoint Configuration = 00 (Non-ISO)
0247 11E1   +1 481      CALL   D12write          ; SoftConnect = 1 (ie attach to USB now)
                                ; Interrupt Mode = 0 (only on good packets)
                                ; Clock running = 1
                                ; No LazyClock = 1
0249 7442   +1 482      MOV     A, #01000010b    ; SOF-ONLY = 0
024B 11E1   +1 483      CALL   D12write          ; Clock Division Factor = 2 == 16MHz
024D       +1 484      InitializeD12DMA:        ; Need to do this AFTER a bus reset
                                ; I do at SetD12Address
024D       +1 485      InitializeInterruptSystem:   ; Initialize the USB level to all OFF
                                ; Now enable the main level
024D E588   +1 486      MOV     A, TimerControl    ; Enable INTO to be level sensitive
024F C2E0   +1 487      CLR     ACC.0
0251 F588   +1 488      MOV     TimerControl, A
0253 75A8C1 +1 489      MOV     EI, #11000001b    ; Enable interrupt subsystem
                                ; INTO for D12 and Ser1 for dScope
0253       +1 490      ; Initialization Complete.
0253       +1 491      ;
0256 00     +1 492      MAIN:  NOP              ; Not much of a main loop for this example
0257 80FD   +1 493      JMP     MAIN              ; All actions are initiated by interrupts
                                ; We are a slave, we wait to be told what to do
0259       +1 494      Wait1msec:          ; A delay loop
0259 758600 +1 495      MOV     DPS, #0          ; Select the primary DPTR
025C 90FB50 +1 496      MOV     DPTR, #-1200
025F A3     +1 497      More:  INC     DPTR        ; 3 cycles
0260 E582   +1 498      MOV     A, DPL          ; + 2
0262 4583   +1 499      ORL    A, DPH          ; + 2
0264 70F9   +1 500      JNZ    More            ; + 3 = 10 cycles x 1200 = 1msec
0266 DFF1   +1 501      DJNZ   R7, Wait1msec
0268 22     +1 502      RET
                                ;
0268       +1 503      ;$include (Decode.A51)
                                ; This module is common to all of the examples.
0268       +1 504      ; It decodes the USB Setup Packets and generates appropriate responses.
0268       +1 505      ; Interpretation of Reports is handled by MAIN
0268       +1 506      ;

```

```

----          +1  521          CSEG
0269          +1  522  ServiceSetupPacket:
0269 E547     +1  523          MOV     A, RequestType
026B A2E7     +1  524          MOV     C, ACC.7          ; Bit 7 = 1 means IO device needs to send
data to P

          C Host
026D 9202     +1  525          MOV     SendData, C
026F 545C     +1  526          ANL    A, #01011100b      ; IF RequestType[6.4.3.2] = 1 THEN goto
BadRequest
0271 7035     +1  527          JNZ    BadRequest
0273 E547     +1  528          MOV     A, RequestType      ; IF RequestType[1&0] = 1 THEN goto
BadRequest
0275 A2E0     +1  529          MOV     C, ACC.0
0277 82E1     +1  530          ANL    C, ACC.1
0279 402D     +1  531          JC     BadRequest
027B 30E502   +1  532          JNB    ACC.5, NotB5        ; IF RequestType[5] = 1 THEN
RequestType[1,0] = [1,
          1]
027E 7403     +1  533          MOV     A, #00000011b
0280 5403     +1  534  NotB5:  ANL    A, #00000011b      ; Set CommandIndex[5,4] = RequestType[1,0]
0282 C4        +1  535          SWAP  A
0283 FF        +1  536          MOV     R7, A              ; Save HI nibble of CommandIndex
          +1  537          ; Set CommandIndex[3,0] = Request[3,0]
0284 E548     +1  538          MOV     A, Request
0286 54F0     +1  539          ANL    A, #111110000b      ; Check if Request > 15
0288 701E     +1  540          JNZ    BadRequest
028A E548     +1  541          MOV     A, Request
028C 540F     +1  542          ANL    A, #00001111b      ; Only 13 are defined today, handle in
table
028E 4F        +1  543          ORL    A, R7
          +1  544          ; CALL  CorrectSubroutine      ; goto CommandTable(CommandIndex)
028F          +1  545  CorrectSubroutine:        ; Jump to the subroutine that DPTR is
pointing to
028F 754F01   +1  546          MOV     ReplyCount, #1      ; Set up a default reply
0292 755000   +1  547          MOV     ReplyBuffer, #0
0295 755100   +1  548          MOV     ReplyBuffer+1, #0
0298 C204     +1  549          CLR    SetAddress          ; Clear all flags
029A C201     +1  550          CLR    STALL
029C C203     +1  551          CLR    IsDescriptor
029E 9002B5   +1  552          MOV     DPTR, #CommandTable
02A1 51AC     +1  553          CALL  BumpDPTR            ; Point to entry
02A3 E0        +1  554          MOVX   A, @DPTR           ; Get the offset
02A4 9002F5   +1  555          MOV     DPTR, #Subroutines
02A7 73        +1  556          JMP     @A+DPTR           ; Go to the correct Subroutine
          +1  557
02A8          +1  558  BadRequest:              ; Decoded a Bad Request, STALL the Endpoint
02A8 D201     +1  559          SETB   STALL
02AA 22        +1  560          RET
          +1  561          ; Support routines
02AB          +1  562  NextDPTR:                ; Returns (DPTR + byte DPTR is pointing to)
02AB E0        +1  563          MOVX   A, @DPTR
02AC          +1  564  BumpDPTR:                ; Returns (DPTR + ACC)
02AC 2582     +1  565          ADD    A, DPL
02AE F582     +1  566          MOV     DPL, A
02B0 5002     +1  567          JNC    Skip
02B2 0583     +1  568          INC    DPH                ; Need 16 bit arithmetic here
02B4 22        +1  569  Skip:  RET
          +1  570
          +1  571          ; Since the table only contains byte offsets, it is important that all these
routines are
          +1  572          ; within one page (100H) of Subroutines
          +1  573          ; V3.0 - CommandTable moved outside of this one page limited space
02B5          +1  574  CommandTable:
          +1  575          ; First 16 commands are for the Device
02B5 16       +1  576          DB LOW(Device_Get_Status - Subroutines)
02B6 00       +1  577          DB LOW(Device_Clear_Feature - Subroutines)
02B7 00       +1  578          DB LOW(Invalid - Subroutines)
02B8 00       +1  579          DB LOW(Device_Set_Feature - Subroutines)
02B9 00       +1  580          DB LOW(Invalid - Subroutines)
02BA 03       +1  581          DB LOW(Set_Address - Subroutines)
02BB 30       +1  582          DB LOW(Get_Descriptor - Subroutines)
02BC 00       +1  583          DB LOW(Set_Descriptor - Subroutines)
02BD 12       +1  584          DB LOW(Get_Configuration - Subroutines)

```

```

02BE 1E      +1  585      DB LOW(Set_Configuration - Subroutines)
02BF 00      +1  586      DB LOW(Invalid - Subroutines)
02C0 00      +1  587      DB LOW(Invalid - Subroutines)
02C1 00      +1  588      DB LOW(Invalid - Subroutines)
02C2 00      +1  589      DB LOW(Invalid - Subroutines)
02C3 00      +1  590      DB LOW(Invalid - Subroutines)
02C4 00      +1  591      DB LOW(Invalid - Subroutines)
              +1  592      ; Next 16 commands are for the Interface
02C5 1A      +1  593      DB LOW(Interface_Get_Status - Subroutines)
02C6 00      +1  594      DB LOW(Interface_Clear_Feature - Subroutines)
02C7 00      +1  595      DB LOW(Invalid - Subroutines)
02C8 00      +1  596      DB LOW(Interface_Set_Feature - Subroutines)
02C9 00      +1  597      DB LOW(Invalid - Subroutines)
02CA 00      +1  598      DB LOW(Invalid - Subroutines)
02CB 57      +1  599      DB LOW(Get_Class_Descriptor - Subroutines)
02CC 00      +1  600      DB LOW(Set_Class_Descriptor - Subroutines)
02CD 00      +1  601      DB LOW(Invalid - Subroutines)
02CE 00      +1  602      DB LOW(Invalid - Subroutines)
02CF 00      +1  603      DB LOW(Get_Interface - Subroutines)
02D0 00      +1  604      DB LOW(Set_Interface - Subroutines)
02D1 00      +1  605      DB LOW(Invalid - Subroutines)
02D2 00      +1  606      DB LOW(Invalid - Subroutines)
02D3 00      +1  607      DB LOW(Invalid - Subroutines)
02D4 00      +1  608      DB LOW(Invalid - Subroutines)
              +1  609      ; Next 16 commands are for the Endpoint
02D5 1A      +1  610      DB LOW(Endpoint_Get_Status - Subroutines)
02D6 00      +1  611      DB LOW(Endpoint_Clear_Feature - Subroutines)
02D7 00      +1  612      DB LOW(Invalid - Subroutines)
02D8 00      +1  613      DB LOW(Endpoint_Set_Feature - Subroutines)
02D9 00      +1  614      DB LOW(Invalid - Subroutines)
02DA 00      +1  615      DB LOW(Invalid - Subroutines)
02DB 00      +1  616      DB LOW(Invalid - Subroutines)
02DC 00      +1  617      DB LOW(Invalid - Subroutines)
02DD 00      +1  618      DB LOW(Invalid - Subroutines)
02DE 00      +1  619      DB LOW(Invalid - Subroutines)
02DF 00      +1  620      DB LOW(Invalid - Subroutines)
02E0 00      +1  621      DB LOW(Invalid - Subroutines)
02E1 00      +1  622      DB LOW(Endpoint_Sync_Frame - Subroutines)
02E2 00      +1  623      DB LOW(Invalid - Subroutines)
02E3 00      +1  624      DB LOW(Invalid - Subroutines)
02E4 00      +1  625      DB LOW(Invalid - Subroutines)
              +1  626      ; Next 16 commands are Class Requests
02E5 00      +1  627      DB LOW(Invalid - Subroutines)
02E6 0B      +1  628      DB LOW(Get_Report - Subroutines)
02E7 00      +1  629      DB LOW(Get_Idle - Subroutines)
02E8 00      +1  630      DB LOW(Get_Protocol - Subroutines)
02E9 00      +1  631      DB LOW(Invalid - Subroutines)
02EA 00      +1  632      DB LOW(Invalid - Subroutines)
02EB 00      +1  633      DB LOW(Invalid - Subroutines)
02EC 00      +1  634      DB LOW(Invalid - Subroutines)
02ED 00      +1  635      DB LOW(Invalid - Subroutines)
02EE 06      +1  636      DB LOW(Set_Report - Subroutines)
02EF 00      +1  637      DB LOW(Set_Idle - Subroutines)
02F0 00      +1  638      DB LOW(Set_Protocol - Subroutines)
02F1 00      +1  639      DB LOW(Invalid - Subroutines)
02F2 00      +1  640      DB LOW(Invalid - Subroutines)
02F3 00      +1  641      DB LOW(Invalid - Subroutines)
02F4 00      +1  642      DB LOW(Invalid - Subroutines)
              +1  643
02F5          +1  644      Subroutines:
              +1  645      ;
              +1  646      ; Many requests are INVALID for this example
02F5          +1  647      Get_Protocol:           ; We are not a Boot device
02F5          +1  648      Set_Protocol:           ; We are not a Boot device
02F5          +1  649      Set_Descriptor:         ; Our Descriptors are static
02F5          +1  650      Set_Class_Descriptor:       ; Our Descriptors are static
  
```

```

02F5      +1 651      Set_Interface:                ; We only have one Interface
02F5      +1 652      Get_Interface:              ; We do not have an Alternate setting
02F5      +1 653      Set_Idle:                  ; Optional command, not supported
02F5      +1 654      Get_Idle:                  ; Optional command, not supported
02F5      +1 655      Device_Set_Feature:        ; We have no features that can be set or cleared
02F5      +1 656      Interface_Set_Feature:    ; We have no features that can be set or cleared
02F5      +1 657      Endpoint_Set_Feature:     ; We have no features that can be set or cleared
02F5      +1 658      Endpoint_Clear_Feature:   ; We have no features that can be set or cleared
02F5      +1 659      Device_Clear_Feature:     ; We have no features that can be set or cleared
02F5      +1 660      Interface_Clear_Feature:  ; We have no features that can be set or cleared
02F5      +1 661      Endpoint_Sync_Frame:      ; We are not an Isonchronous device
02F5      +1 662
02F5      +1 663      Invalid:                    ; Invalid Request made, STALL the Endpoint
02F5 D201  +1 664      SETB      STALL
02F7 22    +1 665      Reply:  RET
02F5      +1 666
02F8      +1 667      Set_Address:                ; Set the address that the SIE will respond to
02F8 D204  +1 668      SETB      SetAddress
02FA 22    +1 669      RET
02F5      +1 670
02FB      +1 671      Set_Report:                  ; Host wants to sent us a Report.
02FB      +1 672      ; The ONLY case in this example where host sends data to us
02FB 3000F7 +1 673      JNB      Configured, Invalid ; Need to be Configured to do this command
02FE 410A   +1 674      JMP      ProcessOutputReport ; RETurn via this subroutine
0300      +1 675      Get_Report:                  ; Host wants a Report
0300 3000F2 +1 676      JNB      Configured, Invalid ; Need to be Configured to do this command
0303 755042 +1 677      MOV      ReplyBuffer, #42H   ; Reply with a recognizable (arbitrary)
value
0306 22    +1 678      RET
0307      +1 679      Get_Configuration:           ; Respond with CurrentConfiguration
0307 854050 +1 680      MOV      ReplyBuffer, CurrentConfiguration
030A 22    +1 681      RET
030B      +1 682      Device_Get_Status:          ; Only two bits of Device Status are
defined
030B 755001 +1 683      MOV      ReplyBuffer, #1      ; Bit 1=Remote Wakeup(=0), Bit 0=Self
Powered(=1)
030E 22    +1 684      RET
030F      +1 685      Interface_Get_Status:       ; Interface Status is currently defined as
0
030F      +1 686      Endpoint_Get_Status:       ;
030F 754F02 +1 687      MOV      ReplyCount, #2      ; Need a two byte 0 response
0312 22    +1 688      RET
0313      +1 689      Set_Configuration:          ; Valid values are 0 and 1
0313 E549   +1 690      MOV      A, wValueLow
0315 6009   +1 691      JZ      Deconfigured
0317 14     +1 692      DEC      A
0318 70DB   +1 693      JNZ      Invalid
031A D200   +1 694      SETB      Configured
031C 754001 +1 695      MOV      CurrentConfiguration, #1
031F 22    +1 696      RET
0320      +1 697      Deconfigured:
0320 C200   +1 698      CLR      Configured
0322 F540   +1 699      MOV      CurrentConfiguration, A
0324 22    +1 700      RET
0325      +1 701      Get_Descriptor:             ; Host wants to know who/what we are
0325 D203   +1 702      SETB      IsDescriptor
0327 E54A   +1 703      MOV      A, wValueHigh
0329 14     +1 704      DEC      A                    ; Valid Values are 1, 2 and 3
032A 900365 +1 705      MOV      DPTR, #DeviceDescriptor
032D 6031   +1 706      JZ      ReturnLength
032F 14     +1 707      DEC      A
0330 900377 +1 708      MOV      DPTR, #ConfigurationDescriptor
0333 7003   +1 709      JNZ      TryString
0335 7422   +1 710      MOV      A, #ConfigLength
0337 22    +1 711      RET
0338      +1 712      TryString:
0338 14     +1 713      DEC      A
0339 70BA   +1 714      JNZ      Invalid
0338      +1 715      ; Request is for a String Descriptor
033B 9003B5 +1 716      MOV      DPTR, #String0          ; Point to String 0

```

```

033E E549      +1  717      MOV      A, wValueLow      ; Get String Index
0340          +1  718      NextString:
0340 601E      +1  719      JZ       ReturnLength
0342 FF       +1  720      MOV      R7, A            ; Save String Index
0343 51AB      +1  721      CALL    NextDPTR
0345 E0       +1  722      MOVX    A, @DPTR         ; Get the String Length (= 0 at Backstop)
0346 60AD      +1  723      JZ       Invalid        ; Asked for a string I don't have
0348 EF       +1  724      MOV      A, R7
0349 14       +1  725      DEC     A
034A 80F4      +1  726      JMP     NextString       ; Check if we are there yet
034C          +1  727      Get_Class_Descriptor:   ; Valid values are 21H, 22H, 23H for Class
Request
034C D203      +1  728      SETB    IsDescriptor
034E E54A      +1  729      MOV      A, wValueHigh
0350 C3       +1  730      CLR     C
0351 9421      +1  731      SUBB    A, #21H
0353 900389    +1  732      MOV      DPTR, #HIDDescriptor
0356 6008      +1  733      JZ       ReturnLength
0358 14       +1  734      DEC     A
0359 900399    +1  735      MOV      DPTR, #ReportDescriptor
035C 6004      +1  736      JZ       ReturnRDlength
035C          +1  737      ; DEC     A            ; This example does not use Physical
Descriptors
035E 8095      +1  738      ; JZ       Send_Physical_Descriptor
035E          +1  739      JMP     Invalid
035E          +1  740      ;
0360          +1  741      ReturnLength:
0360 E0       +1  742      MOVX    A, @DPTR         ; Get Descriptor Length (first byte)
0361 22       +1  743      RET
0362          +1  744      ReturnRDlength:        ; Report Descriptor is different format
0362 741C      +1  745      MOV      A, #ReportLength
0364 22       +1  746      RET
0364          +1  747      ; Error check: this MUST be on within a page of Subroutines:
0070          +1  748      WithinSamePage EQU $ - Subroutines
0364          +1  749      ;
0364          +1  750      ;
0364          +1  751      ;
0364          +1  752      ;$include (DTables.A51)
0364          +1  753      ; This module declares the descriptors
0364          +1  754      ;
0364          +1  755      ; This example has one Device Descriptor with:
0364          +1  756      ; One Configuration - single IN port and single OUT port
0364          +1  757      ; One Interface - there is only one method of accessing the ports
0364          +1  758      ; One HID Descriptor - to make PC host software simpler
0364          +1  759      ; One Endpoint Descriptor - for HID Input Reports
0364          +1  760      ; One Report Descriptor - one byte IN and one byte OUT reports
0364          +1  761      ; Multiple Sting Descriptors - to aid the user
0364          +1  762      ;
----          +1  763      CSEG
0365          +1  764      DeviceDescriptor:
0365 1201      +1  765      DB      18, 1            ; Length, Type
0367 1001      +1  766      DB      10H, 1          ; USB Rev 1.1 (=0110H, low=10H, High=01H)
0369 000000    +1  767      DB      0, 0, 0         ; Class, Subclass and Protocol
036C 10       +1  768      DB      EP0Size        ; EP0 size
036D 4242      +1  769      DW      4242H, 1, 0     ; Vendor ID, Product ID and Version
036F 0001
0371 0000
0373 010200    +1  770      DB      1, 2, 0         ; Manufacturer, Product & Serial# Names
0376 01       +1  771      DB      1              ; #Configs
0377          +1  772      ConfigurationDescriptor:
0377 0902      +1  773      DB      9, 2           ; Length, Type
0379 2200      +1  774      DB      LOW(ConfigLength), HIGH(ConfigLength)
037B 010100    +1  775      DB      1, 1, 0         ; #Interfaces, Configuration#, Config. Name
037E 80       +1  776      DB      10000000b       ; Attributes = Bus Powered
037F FA       +1  777      DB      250            ; Max. Power is 250x2 = 500mA
0380          +1  778      InterfaceDescriptor:
0380 0904      +1  779      DB      9, 4           ; Length, Type
0382 000001    +1  780      DB      0, 0, 1         ; No alternate setting, HID uses EP1

```

```

0385 03      +1 781          DB      3          ; Class = Human Interface Device
0386 0000    +1 782          DB      0, 0       ; Subclass and Protocol
0388 00      +1 783          DB      0          ; Interface Name
0389        +1 784          HIDDescriptor:
0389 0921    +1 785          DB      9, 21H      ; Length, Type
038B 0001    +1 786          DB      0, 1       ; HID Class Specification compliance
038D 00      +1 787          DB      0          ; Country localization (=none)
038E 01      +1 788          DB      1          ; Number of descriptors to follow
038F 22      +1 789          DB      22H      ; And it's a Report descriptor
0390 1C00    +1 790          DB      LOW(ReportLength), HIGH(ReportLength)
0392        +1 791          EndpointDescriptor:
0392 0705    +1 792          DB      7, 5          ; Length, Type
0394 81      +1 793          DB      10000001b    ; Address = IN 1
0395 03      +1 794          DB      00000011b    ; Interrupt
0396 1000    +1 795          DB      EP0Size, 0    ; Maximum packet size (this example only uses 1)
0398 0A      +1 796          DB      10          ; Poll every 10 msec (OS will round down to 8)
      0022    +1 797          ConfigLength EQU $ - ConfigurationDescriptor
      +1 798
0399        +1 799          ReportDescriptor: ; Generated with HID Tool, copied to here
0399 0600FF  +1 800          DB      6, 0, 0FFH ; Usage_Page (Vendor Defined)
039C 0901    +1 801          DB      9, 1          ; Usage (I/O Device)
039E A101    +1 802          DB      0A1H, 1      ; Collection (Application)
03A0 1901    +1 803          DB      19H, 1      ; Usage_Minimum
03A2 2908    +1 804          DB      29H, 8      ; Usage_Maximum
03A4 1500    +1 805          DB      15H, 0      ; Logical_Minimum (0)
03A6 2501    +1 806          DB      25H, 1      ; Logical_Maximum (1)
03A8 7501    +1 807          DB      75H, 1      ; Report_Size (1)
03AA 9508    +1 808          DB      95H, 8      ; Report_Count (8)
03AC 8102    +1 809          DB      81H, 2      ; Input (Data,Var,Abs)
03AE 1901    +1 810          DB      19H, 1      ; Usage_Minimum
03B0 2908    +1 811          DB      29H, 8      ; Usage_Maximum
03B2 9102    +1 812          DB      91H, 2      ; Output (Data,Var,Abs)
03B4 C0      +1 813          DB      0C0H      ; End_Collection
      001C    +1 814          ReportLength EQU $-ReportDescriptor
      +1 815
03B5        +1 816          String0: ; Declare the UNICODE strings
03B5 04030904 +1 817          DB      4, 3, 9, 4 ; Only English language strings supported
03B9        +1 818          String1: ; Manufacturer
03B9 2C03    +1 819          DB      (String2-String1),3 ; Length, Type
03BB 55005300 +1 820          DB      "U",0,"S",0,"B",0," ",0,"D",0,"e",0,"s",0,"i",0,"g",0,"n",0," ",0
03BF 42002000
03C3 44006500
03C7 73006900
03CB 67006E00
03CF 2000
03D1 42007900 +1 821          DB      "B",0,"y",0," ",0,"E",0,"x",0,"a",0,"m",0,"p",0,"l",0,"e",0
03D5 20004500
03D9 78006100
03DD 6D007000
03E1 6C006500
03E5        +1 822          String2: ; Product Name
03E5 1203    +1 823          DB      (EndOfDescriptors-String2),3
03E7 44003100 +1 824          DB      "D",0,"1",0,"2",0," ",0,"T",0,"e",0,"s",0,"t",0
03EB 32002000
03EF 54006500
03F3 73007400
03F7        +1 825          EndOfDescriptors:
03F7 00     +1 826          DB      0          ; Backstop for String Descriptors
      +1 827
      +1 828
      +1 829
      830
      831
      832          END
  
```