

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

LOC	OBJ	LINE	SOURCE
		1	NAME SerialTerminal
		2	;
		3	; Copyright (C) 2001, Intel Corporation
		4	; All rights reserved.
		5	; Permission is hereby granted to merge this program code with other program
		6	; material to create a derivative work. This derivative work may be distributed
		7	; in compiled object form only. Any other publication of this program, in any form,
		8	; without the explicit permission of the copyright holder is prohibited.
		9	;
		10	; Send questions and comments to John.Hyde@intel.com
		11	;
		12	;
		13	; This version works with dScope monSIO0.hex (uses Serial Port 0, loads at 1200H)
		14	;
		15	;\$include (Declare.A51)
	+1	16	; This module declares the variables and constants used in the examples
	+1	17	; It is common to all of the examples
	+1	18	;
	+1	19	; Declare Special Function Registers used
00CD	+1	20	T2High DATA 0CDH
00CC	+1	21	T2Low DATA 0CCH
00CB	+1	22	T2ReloadHigh DATA 0CBH
00CA	+1	23	T2ReloadLow DATA 0CAH
00C8	+1	24	T2Control DATA 0C8H
0098	+1	25	S0Control DATA 098H
0099	+1	26	S0Data DATA 099H
00A8	+1	27	EI DATA 0A8H
00E8	+1	28	EIE DATA 0E8H ; EZ-USB specific
0091	+1	29	EXIF DATA 091H ; EZ-USB specific
00D8	+1	30	EICON DATA 0D8H ; EZ-USB specific
0092	+1	31	PageReg DATA 092H ; EZ-USB specific, used with MOVX @Ri
0086	+1	32	DPS DATA 086H ; EZ-USB specific, used with dual data pointers
	+1	33	;
	+1	34	; "External" memory locations used, EZ-USB specific
	+1	35	; Note that most of these variables are in Page 7FH
7FE8	+1	36	SETUPDAT EQU 07FE8H
7FD4	+1	37	SUDPTR EQU 07FD4H
7FB4	+1	38	EP0Control EQU 07FB4H
7F00	+1	39	EP0InBuffer EQU 07F00H
7EC0	+1	40	EP0OutBuffer EQU 07EC0H ; Not in Page 7FH
7FB6	+1	41	EP1Control EQU 07FB6H
7E80	+1	42	EP1InBuffer EQU 07E80H ; Not in Page 7FH
7FB5	+1	43	IN0ByteCount EQU 07FB5H
7FC5	+1	44	Out0ByteCount EQU 07FC5H
7FB7	+1	45	IN1ByteCount EQU 07FB7H
7FAC	+1	46	IN07IEN EQU 07FACH
7FA9	+1	47	IN07IRQ EQU 07FA9H
7FAD	+1	48	OUT07IEN EQU 07FADH
7FAA	+1	49	OUT07IRQ EQU 07FAAH
7FAE	+1	50	USBIEN EQU 07FAEH
7FAB	+1	51	USBIRQ EQU 07FABH
7FD6	+1	52	USBControl EQU 07FD6H
7FA6	+1	53	I2CData EQU 07FA6H
7FA5	+1	54	I2CControl EQU 07FA5H
7F93	+1	55	PortA_Config EQU 07F93H
7F94	+1	56	PortB_Config EQU 07F94H
7F95	+1	57	PortC_Config EQU 07F95H
7F96	+1	58	PortA_OUT EQU 07F96H

```

7F97      +1  59      PortB_OUT      EQU      07F97H
7F98      +1  60      PortC_OUT      EQU      07F98H
7F99      +1  61      PortA_PINS     EQU      07F99H
7F9A      +1  62      PortB_PINS     EQU      07F9AH
7F9B      +1  63      PortC_PINS     EQU      07F9BH
7F9C      +1  64      PortA_OE      EQU      07F9CH
7F9D      +1  65      PortB_OE      EQU      07F9DH
7F9E      +1  66      PortC_OE      EQU      07F9EH
          +1  67      ;
          +1  68      ; Byte Variables
          +1  69
-----  +1  70
0020      +1  71      FLAGS:          DSEG      AT 20H
          +1  72      ; Bit Variables
          0000      +1  73      Configured     EQU      FLAGS.0 ; Is this device configured
          0001      +1  74      STALL          EQU      FLAGS.1 ; Need to STALL endpoint 0
          0002      +1  75      SendData      EQU      FLAGS.2 ; Need to send data to PC Host
          0003      +1  76      IsDescriptor  EQU      FLAGS.3 ; Enable a shortcut reply
          +1  77      ;
0021      +1  78      MonitorSpace: DS      1FH      ; Used by Dscope
0040      +1  79      Temp:           DS      1        ; A temporary working register
0041      +1  80      Idle_Time:      DS      1        ; The time the PC host wants us to wait
0042      +1  81      Expired_Time:   DS      1        ; A downcounter for timed Reports
0043      +1  82      ReplyBuffer:   DS      3        ; First byte is Count
0046      +1  83      CurrentConfiguration:
0046      +1  84      DS      1        ; Some examples support > 1 configurations
          +1  85      ;
          0004      +1  86      RBFfull      EQU      FLAGS.4 ; Set when the Receive Buffer is FULL
          0005      +1  87      SBFull      EQU      FLAGS.5 ; Set when the Send Buffer is FULL
          0006      +1  88      EP1INReady  EQU      FLAGS.6 ; Set when EP1 IN Ready but no characters to send
          0007      +1  89      TXReady     EQU      FLAGS.7 ; Set when S0 TX is ready but no character to send
0047      +1  90      USBin:         DS      1        ; Head pointer into Send Buffer
0048      +1  91      SerialOUT:     DS      1        ; Tail pointer into Send Buffer
0049      +1  92      SerialIN:      DS      1        ; Head pointer into Receive Buffer
004A      +1  93      USBout:        DS      1        ; Tail pointer into Receive Buffer
004B      +1  94      SendBuffer:    DS      20       ; Buffers characters from USB for the serial port
          005F      +1  95      SendBufferEnd EQU      $
005F      +1  96      ReceiveBuffer: DS      20       ; Buffers characters from the serial port for USB
          0073      +1  97      ReceiveBufferEnd EQU    $
          0028      +1  98      BuffersLength EQU    $-SendBuffer
          +1  99
          +1 100
          +1 101      ;
          +1 102
          +1 103      ;$include (Vectors.A51)
          +1 104      ; This module is common to all of the examples.
          +1 105      ; It contains all of the interrupt vector declarations and
          +1 106      ; the first level interrupt servicing (register save, call subroutine,
          +1 107      ; clear interrupt source, restore registers, return)
          +1 108      ; Suspend and Resume are handled totally in this module
          +1 109      ;
          +1 110      ; A Reset sends us to Program space location 0
-----  +1 111      CSEG AT 0          ; Code space
          +1 112      USING 0          ; Reset forces Register Bank 0
0000 0203E6 +1 113      LJMP      Reset
          +1 114      ;
          +1 115      ; The interrupt vector table is also located here
          +1 116      ; EZ-USB has two levels of USB interrupts:
          +1 117      ; 1-the main level is described in this table (at ORG 43H)
          +1 118      ; 2-there are 21 sources of USB interrupts and these are described in USB_ISR
          +1 119      ; This means that two levels of acknowledgement and clearing will be required
          +1 120      ;      LJMP      INTO_ISR      ; Features not used are commented out
          +1 121      ;      ORG      0BH
          +1 122      ;      LJMP      Timer0_ISR
          +1 123      ;      ORG      13H
          +1 124      ;      LJMP      INT1_ISR

```

```

+1 125 ; ORG 1BH
+1 126 ; LJMP Timer1_ISR
0023 +1 127 ORG 23H
0023 020166 +1 128 LJMP Serial0_ISR
+1 129 ; ORG 2BH
+1 130 ; LJMP Timer2_ISR
+1 131 ; ORG 33H
+1 132 ; LJMP WakeUp_ISR
+1 133 ; ORG 3BH
+1 134 ; LJMP Serial1_ISR ; Used by dScope
0043 +1 135 ORG 43H
0043 020100 +1 136 LJMP USB_ISR ; Auto Vector will replace byte 45H
+1 137 ; ORG 4BH
+1 138 ; LJMP I2C_ISR
+1 139 ; ORG 53H
+1 140 ; LJMP INT4_ISR
+1 141 ; ORG 5BH
+1 142 ; LJMP INT5_ISR
+1 143 ; ORG 63H
+1 144 ; LJMP INT6_ISR
+1 145
00E0 +1 146 ORG 0E0H ; Keep out of the way of dScope monitor
+1 147 ; If you are not using dScope then this memory hole
+1 148 ; may be used for useful routines.
0100 +1 149 ORG 100H
0100 020140 +1 150 USB_ISR:LJMP SUDAV_ISR
0103 00 +1 151 DB 0 ; Pad entries to 4 bytes
0104 02014B +1 152 LJMP SOF_ISR
0107 00 +1 153 DB 0
0108 020124 +1 154 LJMP SUTOK_ISR
010B 00 +1 155 DB 0
010C 020131 +1 156 LJMP Suspend_ISR
010F 00 +1 157 DB 0
0110 02012C +1 158 LJMP USBReset_ISR
0113 00 +1 159 DB 0
0114 020124 +1 160 LJMP Reserved
0117 00 +1 161 DB 0
0118 020124 +1 162 LJMP EP0In_ISR
011B 00 +1 163 DB 0
011C 020124 +1 164 LJMP EP0Out_ISR
011F 00 +1 165 DB 0
0120 020153 +1 166 LJMP EP1In_ISR
0123 00 +1 167 DB 0
+1 168 ; End of Interrupt Vector tables
+1 169
+1 170 ; When a feature is used insert the required interrupt processing here
+1 171 ; The example use only used Endpoints 0 and 1 and also SOF for timing
0124 +1 172 SUTOK_ISR:
0124 +1 173 EP0In_ISR:
0124 +1 174 EP0Out_ISR:
0124 +1 175 Reserved:
0124 32 +1 176 RETI ; Should not get these
+1 177
+1 178 ; Note that MAIN does not use any registers and each interrupt runs to
+1 179 ; completion (no nesting) so there is no need to save registers
+1 180
0125 +1 181 ClearINT2: ; Tell the hardware that we're done
0125 E591 +1 182 MOV A, EXIF
0127 C2E4 +1 183 CLR ACC.4 ; Clear the Interrupt 2 bit
0129 F591 +1 184 MOV EXIF, A
012B 22 +1 185 RET
+1 186
012C +1 187 USBReset_ISR: ; Bus has been Reset, move to DEFAULT state
012C C200 +1 188 CLR Configured
012E 3125 +1 189 CALL ClearINT2
+1 190 ; No need to clear source of interrupt

```

```
0130 32      +1 191          RETI
              +1 192
0131          +1 193      Suspend_ISR:          ; SIE detected an Idle bus
0131 E587    +1 194          MOV      A, PCON
0133 4401    +1 195          ORL      A, #1
0135 F587    +1 196          MOV      PCON, A          ; Go to sleep!
0137 00      +1 197          NOP
0138 00      +1 198          NOP          ; Wake up here due to a USBResume
0139 00      +1 199          NOP
013A 3125    +1 200          CALL     ClearINT2
013C 32      +1 201          RETI
              +1 202
013D          +1 203      WakeUp_ISR:          ; Not using external WAKEUP in these examples
              +1 204          ; So this must be due to a USBResume
013D C2DC    +1 205          CLR      EICON.4          ; Clear the wakeup interrupt source
013F 32      +1 206          RETI
              +1 207
0140          +1 208      SUDAV_ISR:          ; A Setup packet has been received
0140 31D6    +1 209          CALL     ServiceSetupPacket
0142 3125    +1 210          CALL     ClearINT2
              +1 211          ; Clear the source of the interrupt
0144 7401    +1 212          MOV      A, #0000001b
0146 907FAB  +1 213      ExitISR:MOV     DPTR, #USBIRQ
0149 F0      +1 214          MOVX    @DPTR, A
014A 32      +1 215          RETI
              +1 216
014B          +1 217      SOF_ISR:          ; A Start-Of-Frame packet has been received
014B 915E    +1 218          CALL     ServiceTimerRoutine
014D 3125    +1 219          CALL     ClearINT2
              +1 220          ; Clear the source of the interrupt
014F 7402    +1 221          MOV      A, #00000010b
0151 80F3    +1 222          JMP      ExitISR
              +1 223
0153          +1 224      EP1In_ISR:          ; EP1 In Buffer has just been read.
0153 3125    +1 225          CALL     ClearINT2          ; First clear the source of the interrupt
0155 907FA9  +1 226          MOV      DPTR, #IN07IRQ
0158 7402    +1 227          MOV      A, #00000010b
015A F0      +1 228          MOVX    @DPTR, A
              +1 229          ; Create another report if there is a character available
015B A84A    +1 230          MOV      R0, USBout          ; Get pointer into circular buffer
015D E6      +1 231          MOV      A, @R0
015E 7003    +1 232          JNZ     USBCharacterAvailable
0160 D206    +1 233          SETB    EP1INReady          ; Set a flag to say that EP1IN is available
0162 32      +1 234          RETI
0163          +1 235      USBCharacterAvailable:
0163 31BC    +1 236          CALL     CreateInputReport
0165 32      +1 237          RETI
              +1 238
0166          +1 239      Serial0_ISR:
0166 E598    +1 240          MOV      A, S0Control          ; Check for TXDone or RXReady
0168 20E119  +1 241          JB      ACC.1, TXDone
016B          +1 242      RXReady:          ; Character just received
016B C298    +1 243          CLR     S0Control.0          ; Clear the Interrupt bit
016D A849    +1 244          MOV      R0, SerialIN          ; Is there room in the Receive Buffer?
016F E6      +1 245          MOV      A, @R0
0170 6003    +1 246          JZ      OKtoWriteRB
0172 D204    +1 247          SETB    RBFULL
0174 32      +1 248          RETI
0175          +1 249      OKtoWriteRB:
0175 E599    +1 250          MOV      A, S0Data          ; Get the character
0177 F6      +1 251          MOV      @R0, A          ; And put it in the Receive Buffer
0178 08      +1 252          INC     R0          ; Bump the buffer pointer
0179 B87302  +1 253          CJNE   R0, #ReceiveBufferEnd, NoWrapRB1
017C 785F    +1 254          MOV      R0, #ReceiveBuffer
017E          +1 255      NoWrapRB1:
017E 8849    +1 256          MOV      SerialIN, R0
```

```
+1 257 ;
+1 258 ; Check to see if USB is waiting for a character
0180 2006E0 +1 259 JB EPLINReady, USBCharacterAvailable
0183 32 +1 260 RETI
+1 261
0184 +1 262 TXDone: ; Just finished transmitting a character
0184 C299 +1 263 CLR S0Control.1 ; Clear the Interrupt bit
0186 A848 +1 264 MOV R0, SerialOUT
0188 E6 +1 265 MOV A, @R0 ; Is another available
0189 7003 +1 266 JNZ SerialCharacterAvailable
018B D207 +1 267 SETB TXReady
018D 32 +1 268 RETI
018E +1 269 SerialCharacterAvailable:
018E 3191 +1 270 CALL SendChar
0190 32 +1 271 RETI
+1 272
0191 +1 273 SendChar: ; Send a character to the serial port
0191 A848 +1 274 MOV R0, SerialOUT
0193 E6 +1 275 MOV A, @R0
0194 F599 +1 276 MOV S0Data, A
0196 E4 +1 277 CLR A
0197 F6 +1 278 MOV @R0, A ; Clear the character entry just sent
0198 08 +1 279 INC R0
0199 B85F02 +1 280 CJNE R0, #SendBufferEnd, NoWrapSB1
019C 784B +1 281 MOV R0, #SendBuffer
019E +1 282 NoWrapSB1:
019E 8848 +1 283 MOV SerialOUT, R0
01A0 C207 +1 284 CLR TXReady
01A2 22 +1 285 RET
+1 286
+1 287 ; ProcessOutputReport and CreateInputReport moved into Vectors module so
+1 288 ; that filling/emptying the Send/Receive buffers was in one place
+1 289
01A3 +1 290 ProcessOutputReport: ; A Report has just been received
+1 291 ; The report contains a character to be sent to the Serial port
+1 292 ; Save the character in the Send circular buffer
01A3 907EC0 +1 293 MOV DPTR, #EP0OutBuffer ; Point to the Report
01A6 A847 +1 294 MOV R0, USBin ; Point into the Send Buffer
01A8 E6 +1 295 MOV A, @R0 ; Get the entry in the Send Buffer
01A9 6003 +1 296 JZ OKtoWriteSB ; Check that it's empty
01AB D205 +1 297 SETB SBFull ; Set an error bit
01AD 22 +1 298 RET ; Throw away the newest characters!
01AE +1 299 OKtoWriteSB:
01AE E0 +1 300 MOVX A, @DPTR ; Get the character
01AF F6 +1 301 MOV @R0, A ; Write to the Send Buffer
01B0 08 +1 302 INC R0 ; Bump the buffer pointer
01B1 B85F02 +1 303 CJNE R0, #SendBufferEnd, NoWrapSB2
01B4 784B +1 304 MOV R0, #SendBuffer ; Maintain a circular buffer
01B6 +1 305 NoWrapSB2:
01B6 8847 +1 306 MOV USBin, R0 ; Save IN pointer
+1 307 ;
+1 308 ; Check to see if the Serial port is waiting for a character
01B8 2007D6 +1 309 JB TXReady, SendChar
01BB 22 +1 310 RET
+1 311
01BC +1 312 CreateInputReport:
+1 313 ; The report contains a character from the serial port
+1 314 ; Get the character from the Receive circular buffer
01BC 907E80 +1 315 MOV DPTR, #EP1InBuffer ; Point to the buffer
01BF A84A +1 316 MOV R0, USBout ; Point into the Receive Buffer
01C1 E6 +1 317 MOV A, @R0 ; Get character from Receive Buffer
01C2 F0 +1 318 MOVX @DPTR, A ; and update the Report
01C3 E4 +1 319 CLR A
01C4 F6 +1 320 MOV @R0, A ; Clear the Receive Buffer entry just read
01C5 08 +1 321 INC R0
01C6 B87302 +1 322 CJNE R0, #ReceiveBufferEnd, NoWrapRB2
```

```
01C9 785F      +1 323      MOV      R0, #ReceiveBuffer      ; Maintain a circular buffer
01CB          +1 324      NoWrapRB2:
01CB 884A      +1 325      MOV      USBout, R0              ; Save OUT pointer
01CD 907FB7    +1 326      MOV      DPTR, #IN1ByteCount
01D0 7401      +1 327      MOV      A, #1
01D2 F0       +1 328      MOVX     @DPTR, A                ; Endpoint 1 now 'armed', next IN will get
data
01D3 C206      +1 329      CLR      EP1INReady
01D5 22       +1 330      RET
+1 331
+1 332
+1 333      ;$include (USB_INT.A51)
+1 334      ; This module is common to all of the examples.
+1 335      ; It services USB Requests from the SIE.
+1 336      ; Interpretation of the Output Reports is handled by MAIN
+1 337      ;
---- +1 338      CSEG
01D6          +1 339      ServiceSetupPacket:
01D6 907FE8    +1 340      MOV      DPTR, #SETUPDAT        ; Point to Setup Packet data
01D9 E0       +1 341      MOVX     A, @DPTR              ; Get the RequestType
01DA A2E7     +1 342      MOV      C, ACC.7             ; Bit 7 = 1 means IO device needs to send
data to P
C Host
01DC 9202     +1 343      MOV      SendData, C
01DE 545C     +1 344      ANL      A, #01011100b        ; IF RequestType[6.4.3.2] = 1 THEN goto
BadRequest
01E0 7050     +1 345      JNZ      BadRequest
01E2 E0       +1 346      MOVX     A, @DPTR              ; IF RequestType[1&0] = 1 THEN goto
BadRequest
01E3 A2E0     +1 347      MOV      C, ACC.0
01E5 82E1     +1 348      ANL      C, ACC.1
01E7 4049     +1 349      JC       BadRequest
01E9 30E502   +1 350      JNB      ACC.5, NotB5         ; IF RequestType[5] = 1 THEN
RequestType[1,0] = [1,
1]
01EC 7403     +1 351      MOV      A, #00000011b
01EE 5403     +1 352      NotB5:  ANL      A, #00000011b    ; Set CommandIndex[5,4] = RequestType[1,0]
01F0 C4       +1 353      SWAP     A
01F1 F540     +1 354      MOV      Temp, A              ; Save HI nibble of CommandIndex
+1 355      ; Set CommandIndex[3,0] = Request[3,0]
01F3 A3       +1 356      INC      DPTR                  ; Point to Request
01F4 E0       +1 357      MOVX     A, @DPTR
01F5 540F     +1 358      ANL      A, #00001111b        ; Only 13 are defined today, handle in
table
01F7 4540     +1 359      ORL      A, Temp
01F9 5141     +1 360      CALL    CorrectSubroutine     ; goto CommandTable(CommandIndex)
+1 361      ; Returns STALL=1 if a stall is required
01FB 200134   +1 362      JB       STALL, BadRequest
01FE 300218   +1 363      JNB      SendData, HandShake
0201 200320   +1 364      JB       IsDescriptor, LoadSUDPTR; EZ-USB has a short cut for descriptors
+1 365      ; Send data in ReplyBuffer
0204 907F02   +1 366      MOV      DPTR, #EP0InBuffer+2
0207 7846     +1 367      MOV      R0, #ReplyBuffer+3
0209 754003   +1 368      MOV      Temp, #3              ; Copy maximum byte count
020C E6       +1 369      CopyRB: MOV      A, @R0
020D F0       +1 370      MOVX     @DPTR, A
020E 1582     +1 371      DEC      DPL
0210 18       +1 372      DEC      R0
0211 D540F8   +1 373      DJNZ     Temp, CopyRB
0214 E6       +1 374      MOV      A, @R0                ; Get real byte count
0215          +1 375      SendEP0InBuffer:
0215 907FB5   +1 376      MOV      DPTR, #In0ByteCount
0218          +1 377      StartXfer:
0218 F0       +1 378      MOVX     @DPTR, A              ; This write initiates the transfer
0219          +1 379      HandShake:
0219 754002   +1 380      MOV      Temp, #00000010b     ; Set HSNACK to tell the SIE that we're done
021C          +1 381      SetEP0Control:
021C 907FB4   +1 382      MOV      DPTR, #EP0Control
021F E0       +1 383      MOVX     A, @DPTR
0220 4540     +1 384      ORL      A, Temp
0222 F0       +1 385      MOVX     @DPTR, A
0223 22       +1 386      RET
```

```

0224      +1 387      LoadSUDPTR:                                ; Send the data pointed to by DPTR
0224 858240 +1 388          MOV      Temp, DPL
0227 E583   +1 389          MOV      A, DPH
0229 907FD4 +1 390          MOV      DPTR, #SUDPTR
022C F0     +1 391          MOVX    @DPTR, A
022D E540   +1 392          MOV      A, Temp
022F A3     +1 393          INC      DPTR
0230 80E6   +1 394          JMP      StartXfer
0232      +1 395      BadRequest:                            ; Invalid Request was received
0232 754003 +1 396          MOV      Temp, #0000011b          ; Set EPOSTALL and HSNACK
0235 80E5   +1 397          JMP      SetEP0Control
                                +1 398
0237      +1 399      NextDPTR:                                ; Returns (DPTR + byte DPTR is pointing to)
0237 E0     +1 400          MOVX    A, @DPTR
0238      +1 401      BumpDPTR:                                ; Returns (DPTR + ACC)
0238 2582   +1 402          ADD      A, DPL
023A F582   +1 403          MOV      DPL, A
023C 5002   +1 404          JNC     Skip
023E 0583   +1 405          INC     DPH                                ; Need 16 bit arithmetic here
0240 22     +1 406      Skip:      RET
                                +1 407
0241      +1 408      CorrectSubroutine:                    ; Jump to the subroutine that DPTR is
pointing to
0241 900266 +1 409          MOV      DPTR, #CommandTable
0244 5138   +1 410          CALL   BumpDPTR                                ; Point to entry
0246 E0     +1 411          MOVX    A, @DPTR                                ; Get the offset
0247 900266 +1 412          MOV      DPTR, #CommandTable
024A 5138   +1 413          CALL   BumpDPTR                                ; Get the routine address
024C C082   +1 414          PUSH   DPL                                ; Create a RETURN address on stack
024E C083   +1 415          PUSH   DPH                                ; Note: JMP @A+DPTR not used since A, DPTR
needed
0250 7845   +1 416          MOV      R0, #ReplyBuffer+2
0252 E4     +1 417          CLR     A
0253 F6     +1 418          MOV     @R0, A                                ; Clear ReplyBuffer
0254 18     +1 419          DEC     R0
0255 F6     +1 420          MOV     @R0, A
0256 18     +1 421          DEC     R0
0257 7601   +1 422          MOV     @R0, #1                                ; Default non-descriptor reply
0259 907FEA +1 423          MOV     DPTR, #SETUPDAT+2          ; Point to LOW(wValue)
025C E0     +1 424          MOVX    A, @DPTR                                ; Many of the routines need these
025D F5F0   +1 425          MOV     B, A                                ; LOW(wValue) in B
025F A3     +1 426          INC     DPTR
0260 E0     +1 427          MOVX    A, @DPTR                                ; HIGH(wValue) in A
0261 C201   +1 428          CLR     STALL
0263 C203   +1 429          CLR     IsDescriptor
0265 22     +1 430          RET
                                +1 431
routines are +1 432      ; Since the table only contains byte offsets, it is important that all these
                                +1 433      ; within one page (100H) of CommandTable
                                +1 434      ;
0266      +1 435      CommandTable:
                                +1 436      ; First 16 commands are for the Device
0266 6B     +1 437          DB     Device_Get_Status - CommandTable
0267 40     +1 438          DB     Device_Clear_Feature - CommandTable
0268 40     +1 439          DB     Invalid - CommandTable
0269 40     +1 440          DB     Device_Set_Feature - CommandTable
026A 40     +1 441          DB     Invalid - CommandTable
026B 40     +1 442          DB     Invalid - CommandTable                                ; SIE implements Device_Set_Address
026C 95     +1 443          DB     Get_Descriptor - CommandTable
026D 40     +1 444          DB     Set_Descriptor - CommandTable
026E 68     +1 445          DB     Get_Configuration - CommandTable
026F 72     +1 446          DB     Set_Configuration - CommandTable
0270 40     +1 447          DB     Invalid - CommandTable
0271 40     +1 448          DB     Invalid - CommandTable
0272 40     +1 449          DB     Invalid - CommandTable
0273 40     +1 450          DB     Invalid - CommandTable
0274 40     +1 451          DB     Invalid - CommandTable
0275 40     +1 452          DB     Invalid - CommandTable

```

```

+1 453 ; Next 16 commands are for the Interface
0276 6F +1 454 DB Interface_Get_Status - CommandTable
0277 40 +1 455 DB Interface_Clear_Feature - CommandTable
0278 40 +1 456 DB Invalid - CommandTable
0279 40 +1 457 DB Interface_Set_Feature - CommandTable
027A 40 +1 458 DB Invalid - CommandTable
027B 40 +1 459 DB Invalid - CommandTable
027C B9 +1 460 DB Get_Class_Descriptor - CommandTable
027D 40 +1 461 DB Set_Class_Descriptor - CommandTable
027E 40 +1 462 DB Invalid - CommandTable
027F 40 +1 463 DB Invalid - CommandTable
0280 40 +1 464 DB Get_Interface - CommandTable
0281 40 +1 465 DB Set_Interface - CommandTable
0282 40 +1 466 DB Invalid - CommandTable
0283 40 +1 467 DB Invalid - CommandTable
0284 40 +1 468 DB Invalid - CommandTable
0285 40 +1 469 DB Invalid - CommandTable
+1 470 ; Next 16 commands are for the Endpoint
0286 6F +1 471 DB Endpoint_Get_Status - CommandTable
0287 42 +1 472 DB Endpoint_Clear_Feature - CommandTable
0288 40 +1 473 DB Invalid - CommandTable
0289 40 +1 474 DB Endpoint_Set_Feature - CommandTable
028A 40 +1 475 DB Invalid - CommandTable
028B 40 +1 476 DB Invalid - CommandTable
028C 40 +1 477 DB Invalid - CommandTable
028D 40 +1 478 DB Invalid - CommandTable
028E 40 +1 479 DB Invalid - CommandTable
028F 40 +1 480 DB Invalid - CommandTable
0290 40 +1 481 DB Invalid - CommandTable
0291 40 +1 482 DB Invalid - CommandTable
0292 40 +1 483 DB Endpoint_Sync_Frame - CommandTable
0293 40 +1 484 DB Invalid - CommandTable
0294 40 +1 485 DB Invalid - CommandTable
0295 40 +1 486 DB Invalid - CommandTable
+1 487 ; Next 16 commands are Class Requests
0296 40 +1 488 DB Invalid - CommandTable
0297 54 +1 489 DB Get_Report - CommandTable
0298 61 +1 490 DB Get_Idle - CommandTable
0299 40 +1 491 DB Get_Protocol - CommandTable
029A 40 +1 492 DB Invalid - CommandTable
029B 40 +1 493 DB Invalid - CommandTable
029C 40 +1 494 DB Invalid - CommandTable
029D 40 +1 495 DB Invalid - CommandTable
029E 40 +1 496 DB Invalid - CommandTable
029F 43 +1 497 DB Set_Report - CommandTable
02A0 5B +1 498 DB Set_Idle - CommandTable
02A1 40 +1 499 DB Set_Protocol - CommandTable
02A2 40 +1 500 DB Invalid - CommandTable
02A3 40 +1 501 DB Invalid - CommandTable
02A4 40 +1 502 DB Invalid - CommandTable
02A5 40 +1 503 DB Invalid - CommandTable
+1 504 ;
+1 505 ; Many requests are INVALID for this example
02A6 +1 506 Get_Protocol: ; We are not a Boot device
02A6 +1 507 Set_Protocol: ; We are not a Boot device
02A6 +1 508 Set_Descriptor: ; Our Descriptors are static
02A6 +1 509 Set_Class_Descriptor: ; Our Descriptors are static
02A6 +1 510 Set_Interface: ; We only have one Interface
02A6 +1 511 Get_Interface: ; We do not have an Alternate setting
02A6 +1 512 Device_Set_Feature: ; We have no features that can be set or cleared
02A6 +1 513 Interface_Set_Feature: ; We have no features that can be set or cleared
02A6 +1 514 Endpoint_Set_Feature: ; We have no features that can be set or cleared
02A6 +1 515 Device_Clear_Feature: ; We have no features that can be set or cleared
02A6 +1 516 Interface_Clear_Feature: ; We have no features that can be set or cleared
02A6 +1 517 Endpoint_Sync_Frame: ; We are not an Isonchronous device
+1 518

```

```
02A6      +1 519      Invalid:                ; Invalid Request made, STALL the Endpoint
02A6 D201  +1 520      SETB     STALL
           +1 521      ;
02A8      +1 522      Endpoint_Clear_Feature: ; We have no features that can be set or cleared
           +1 523      ;
02A8 22    +1 524      RET
           +1 525
02A9      +1 526      Set_Report:                ; Host wants to sent us a Report.
           +1 527      ; The ONLY case in this example where host sends data to us
02A9 3000FA +1 528      JNB     Configured, Invalid ; Need to be Configured to do this command
02AC 907FC5 +1 529      MOV     DPTR, #Out0ByteCount ; Enable EP0OutBuffer to receive data
02AF F0     +1 530      MOVX    @DPTR, A           ; Any value will do, this sets OUTBSY
02B0 907FB4 +1 531      MOV     DPTR, #EP0Control   ; Wait for valid data in EP0OutBuffer
02B3 E0     +1 532      Wait40: MOVX   A, @DPTR
02B4 5408   +1 533      ANL     A, #00001000b      ; Check OUTBSY
02B6 70FB   +1 534      JNZ     Wait40
02B8 21A3   +1 535      JMP     ProcessOutputReport ; RETURN via this subroutine
02BA      +1 536      Get_Report:                ; Host wants a Report
02BA 3000E9 +1 537      JNB     Configured, Invalid ; Need to be Configured to do this command
02BD 08     +1 538      INC     R0                ; Point to ReplyBuffer(1)
02BE 7618   +1 539      MOV     @R0, #18H          ; Reply with a recognizable (arbitrary)
value
02C0 22    +1 540      Reply: RET
02C1      +1 541      Set_Idle:                ; Host wants to tell us how often we should
talk
02C1 3000E2 +1 542      JNB     Configured, Invalid ; Need to be Configured to do this command
02C4 F541   +1 543      MOV     Idle_Time, A
02C6 22    +1 544      RET
           +1 545      Get_Idle:                ; Handshake with host
           +1 545      ; Host must have forgotten what he told us
to do
02C7 3000DC +1 546      JNB     Configured, Invalid ; Need to be Configured to do this command
02CA 08     +1 547      INC     R0                ; Point to ReplyBuffer(1)
02CB A641   +1 548      MOV     @R0, Idle_Time
02CD 22    +1 549      RET
02CE      +1 550      Get_Configuration:          ; Need to return 0 or 1
02CE 300004 +1 551      JNB     Configured, Configuration0
02D1      +1 552      Configuration1:          ; Same bit pattern as Device_Get_Status
02D1      +1 553      Device_Get_Status:        ; Only two bits of Device Status are
defined
02D1 08     +1 554      INC     R0                ; Point to ReplyBuffer(1)
02D2 7601   +1 555      MOV     @R0, #1           ; Bit 1=Remote Wakeup(=0), Bit 0=Self
Powered(=1)
02D4 22    +1 556      RET
02D5      +1 557      Configuration0:          ; Same bit pattern as Interface_Get_Status
02D5      +1 558      Interface_Get_Status:      ; Interface Status is currently defined as
0
02D5      +1 559      Endpoint_Get_Status:
02D5 7602   +1 560      MOV     @R0, #2
02D7 22    +1 561      RET
02D8      +1 562      Set_Configuration:          ; Valid values are 0 and 1
02D8 E5F0   +1 563      MOV     A, B                ; Get LOW(wValue)
02DA 6011   +1 564      JZ     Deconfigured
02DC 14     +1 565      DEC     A
02DD 70C7   +1 566      JNZ     Invalid
02DF D200   +1 567      SETB    Configured
           +1 568      ; It is now OK to create input reports.
           +1 569      ; Enable Endpoint 1 IN interrupts
02E1 907FAC +1 570      MOV     DPTR, #IN07IEN          ; Get interrupt mask
02E4 E0     +1 571      MOVX   A, @DPTR
02E5 4402   +1 572      ORL     A, #00000010b      ; Set bit 1 = IN1IEN
02E7 F0     +1 573      MOVX   @DPTR, A
02E8 D206   +1 574      SETB    EPLINReady
02EA D207   +1 575      SETB    TXReady          ; Kick start the Serial Port
02EC 22    +1 576      RET
02ED      +1 577      Deconfigured:
02ED C200   +1 578      CLR     Configured
           +1 579      ; Disable Input Report Generation
02EF 907FAC +1 580      MOV     DPTR, #IN07IEN          ; Get interrupt mask
02F2 E0     +1 581      MOVX   A, @DPTR
02F3 54FD   +1 582      ANL     A, #11111101b      ; Set bit 1 = 0 (disable)
02F5 F0     +1 583      MOVX   @DPTR, A
02F6 C206   +1 584      CLR     EPLINReady
```

```

02F8 C207      +1  585          CLR    TXReady
02FA 22        +1  586          RET
02FB          +1  587      Get_Descriptor:                ; Host wants to know who/what we are
02FB D203      +1  588          SETB   IsDescriptor
02FD 14        +1  589          DEC    A                        ; Valid Values are 1, 2 and 3
02FE 900345    +1  590          MOV    DPTR, #DeviceDescriptor
0301 60BD      +1  591          JZ     Reply
0303 14        +1  592          DEC    A
0304 900357    +1  593          MOV    DPTR, #ConfigurationDescriptor
0307 60B7      +1  594          JZ     Reply
0309 14        +1  595          DEC    A
030A 709A      +1  596          JNZ   Invalid
              +1  597      ; Request is for a String Descriptor
030C 900395    +1  598          MOV    DPTR, #String0          ; Point to String 0
030F E5F0      +1  599          MOV    A, B                  ; Get String Index
0311          +1  600      NextString:
0311 601E      +1  601          JZ     FixUpthenReply
0313 F540      +1  602          MOV    Temp, A              ; Save String Index
0315 5137      +1  603          CALL  NextDPTR
0317 E0        +1  604          MOVX   A, @DPTR             ; Get the String Length (= 0 means we're at
Backsto
              p)
0318 608C      +1  605          JZ     Invalid              ; Asked for a string I don't have
031A E540      +1  606          MOV    A, Temp
031C 14        +1  607          DEC    A
031D 80F2      +1  608          JMP    NextString          ; Check if we are there yet
031F          +1  609      Get_Class_Descriptor:        ; Valid values are 21H, 22H, 23H for Class
Request
031F D203      +1  610          SETB   IsDescriptor
0321 C3        +1  611          CLR    C
0322 9421      +1  612          SUBB   A, #21H
0324 900369    +1  613          MOV    DPTR, #HIDDescriptor
0327 6097      +1  614          JZ     Reply
0329 14        +1  615          DEC    A
032A 900379    +1  616          MOV    DPTR, #ReportDescriptor
032D 6091      +1  617          JZ     Reply
              +1  618      ; DEC    A                        ; This example does not use Physical
Descriptors
              +1  619      ; JZ     Send_Physical_Descriptor
032F 41A6      +1  620          JMP    Invalid
              +1  621      ;
              +1  622      ; Error check: this MUST be on within a page of CommandTable
00CB          +1  623      WithinSamePage EQU $ - CommandTable
              +1  624      ;
0331          +1  625      FixUpthenReply:                ; EZ-USB Rev D has a String Descriptor bug
              +1  626          ; Need to fill the IN0BUF (@ 7F00H) myself
0331 E0        +1  627          MOVX   A, @DPTR             ; Get the string length
0332 FF        +1  628          MOV    R7, A              ; Save counter
0333 F5F0      +1  629          MOV    B, A
0335 7800      +1  630          MOV    R0, #LOW(EP0InBuffer) ; PageReg = 7FH = HIGH(EP0InBuffer)
0337 F2        +1  631      CopySD: MOVX   @R0, A
0338 08        +1  632          INC    R0
0339 A3        +1  633          INC    DPTR
033A E0        +1  634          MOVX   A, @DPTR
033B DFFA      +1  635          DJNZ  R7, CopySD
              +1  636      ; Fixup complete, get back to the program flow
033D D0E0      +1  637          POP    ACC                ; Get rid of the return address
033F D0E0      +1  638          POP    ACC
0341 E5F0      +1  639          MOV    A, B              ; Retrieve byte count
0343 4115      +1  640          JMP    SendEP0InBuffer
              641
              642      ;$include (DTables.A51)
              +1  643      ; This module declares the descriptors
              +1  644      ;
              +1  645      ; This example has one Device Descriptor with:
              +1  646      ;     One Configuration - single IN port and single OUT port
              +1  647      ;     One Interface - there is only one method of accessing the ports
              +1  648      ;     One HID Descriptor - to make PC host software simpler
              +1  649      ;     One Endpoint Descriptor - for HID Input Reports

```

```

+1 650 ; One Report Descriptor - one byte IN and one byte OUT reports
+1 651 ; Multiple Sting Descriptors - to aid the user
+1 652 ;
----
+1 653 CSEG
0345 +1 654 DeviceDescriptor:
0345 1201 +1 655 DB 18, 1 ; Length, Type
0347 0101 +1 656 DW 101H ; USB Rev 1.1
0349 000000 +1 657 DB 0, 0, 0 ; Class, Subclass and Protocol
034C 40 +1 658 DB 64 ; EP0 size
034D 4242 +1 659 DW 4242H, 1, 1 ; Vendor ID, Product ID and Version
034F 0001
0351 0001
0353 010200 +1 660 DB 1, 2, 0 ; Manufacturer, Product & Serial# Names
0356 01 +1 661 DB 1 ; #Configs
0357 +1 662 ConfigurationDescriptor:
0357 0902 +1 663 DB 9, 2 ; Length, Type
0359 2200 +1 664 DB LOW(ConfigLength), HIGH(ConfigLength)
035B 010100 +1 665 DB 1, 1, 0 ; #Interfaces, Configuration#, Config. Name
035E 80 +1 666 DB 10000000b ; Attributes = Bus Powered
035F 32 +1 667 DB 50 ; Max. Power is 50x2 = 100mA
0360 +1 668 InterfaceDescriptor:
0360 0904 +1 669 DB 9, 4 ; Length, Type
0362 000001 +1 670 DB 0, 0, 1 ; No alternate setting, HID uses EP1
0365 03 +1 671 DB 3 ; Class = Human Interface Device
0366 0000 +1 672 DB 0, 0 ; Subclass and Protocol
0368 00 +1 673 DB 0 ; Interface Name
0369 +1 674 HIDDescriptor:
0369 0921 +1 675 DB 9, 21H ; Length, Type
036B 0001 +1 676 DB 0, 1 ; HID Class Specification compliance
036D 00 +1 677 DB 0 ; Country localization (=none)
036E 01 +1 678 DB 1 ; Number of descriptors to follow
036F 22 +1 679 DB 22H ; And it's a Report descriptor
0370 1C00 +1 680 DB LOW(ReportLength), HIGH(ReportLength)
0372 +1 681 EndpointDescriptor:
0372 0705 +1 682 DB 7, 5 ; Length, Type
0374 81 +1 683 DB 10000001b ; Address = IN 1
0375 03 +1 684 DB 00000011b ; Interrupt
0376 4000 +1 685 DB 64, 0 ; Maximum packet size (this example only uses 1)
0378 0A +1 686 DB 10 ; Poll every 100 msec
0022 +1 687 ConfigLength EQU $ - ConfigurationDescriptor
+1 688
0379 +1 689 ReportDescriptor: ; Generated with HID Tool, copied to here
0379 0600FF +1 690 DB 6, 0, 0FFH ; Usage_Page (Vendor Defined)
037C 0901 +1 691 DB 9, 1 ; Usage (I/O Device)
037E A101 +1 692 DB 0A1H, 1 ; Collection (Application)
0380 1901 +1 693 DB 19H, 1 ; Usage_Minimum
0382 2902 +1 694 DB 29H, 2 ; Usage_Maximum
0384 1500 +1 695 DB 15H, 0 ; Logical_Minimum (0)
0386 2501 +1 696 DB 25H, 1 ; Logical_Maximum (1)
0388 7508 +1 697 DB 75H, 8 ; Report_Size (8)
038A 9501 +1 698 DB 95H, 1 ; Report_Count (1)
038C 8102 +1 699 DB 81H, 2 ; Input (Data,Var,Abs)
038E 1901 +1 700 DB 19H, 1 ; Usage_Minimum
0390 2902 +1 701 DB 29H, 2 ; Usage_Maximum
0392 9102 +1 702 DB 91H, 2 ; Output (Data,Var,Abs)
0394 C0 +1 703 DB 0C0H ; End_Collection
001C +1 704 ReportLength EQU $ -ReportDescriptor
+1 705
0395 +1 706 String0: ; Declare the UNICODE strings
0395 04030904 +1 707 DB 4, 3, 9, 4 ; Only English language strings supported
0399 +1 708 String1: ; Manufacturer
0399 2C03 +1 709 DB (String2-String1),3 ; Length, Type
039B 55005300 +1 710 DB "U",0,"S",0,"B",0," ",0,"D",0,"e",0,"s",0,"i",0,"g",0,"n",0," ",0
039F 42002000
03A3 44006500
03A7 73006900

```

```

03AB 67006E00
03AF 2000
03B1 42007900 +1 711          DB      "B",0,"y",0," ",0,"E",0,"x",0,"a",0,"m",0,"p",0,"l",0,"e",0
03B5 20004500
03B9 78006100
03BD 6D007000
03C1 6C006500
03C5          +1 712      String2:          ; Product Name
03C5 1003      +1 713          DB      (EndOfDescriptors-String2),3
03C7 53006500 +1 714          DB      "S",0,"e",0,"r",0,"i",0,"a",0,"l",0,"l",0
03CB 72006900
03CF 61006C00
03D3 3100
03D5          +1 715      EndOfDescriptors:
03D5 0000      +1 716          DW      0          ; Backstop for String Descriptors
          +1 717
          +1 718
          +1 719
          720
          721      ;$include (Main.A51)
          +1 722      ; This module initializes the microcontroller then executes MAIN forever
          +1 723      ;
03D7 0D0D5365 +1 724      SignOnMessage: DB 13, 13, "Serial1 V0.9", 13 ;USB Serial Device
03DB 7269616C
03DF 31205630
03E3 2E390D
          000F      +1 725      SignOnLength EQU $-SignOnMessage
          +1 726
03E6          +1 727      Reset:
03E6 7581EB      +1 728          MOV      SP, #235          ; Initialize the Stack
03E9 75927F      +1 729          MOV      PageReg, #7FH      ; Allows MOVX Ri to access EZ-USB memory
          +1 730
03EC 78D6          +1 731          MOV      R0, #Low(USBControl) ; Simulate a disconnect
03EE E2          +1 732          MOVX     A, @R0
03EF 54F3          +1 733          ANL     A, #11110011b      ; Clear DISCON, DISCOE
03F1 F2          +1 734          MOVX     @R0, A
03F2 914D          +1 735          CALL   Wait100msec        ; Give the host time to react
03F4 E2          +1 736          MOVX     A, @R0          ; Reconnect with this new identity
03F5 4406          +1 737          ORL     A, #00000110b    ; Set DISCOE to enable pullup resistor
03F7 F2          +1 738          MOVX     @R0, A          ; Set RENUM so that 8051 handles USB
requests
03F8 E4          +1 739          CLR     A
03F9 F520          +1 740          MOV     FLAGS, A        ; Start in Default state
03FB          +1 741      InitializeVariables:
03FB 784B          +1 742          MOV     R0, #SendBuffer
03FD AF28          +1 743          MOV     R7, BuffersLength
03FF F6          +1 744      IVLoop: MOV     @R0, A        ; Clear Send and Receive Buffers
0400 08          +1 745          INC     R0
0401 DFFC          +1 746          DJNZ   R7, IVLoop
          +1 747
0403 9003D7       +1 748          MOV     DPTR, #SignOnMessage ; Put a Signon in the Receive Buffer
0406 7F0F          +1 749          MOV     R7, #SignonLength
0408 785F          +1 750          MOV     R0, #ReceiveBuffer
040A E0          +1 751      IVLoop2:MOVX   A, @DPTR
040B F6          +1 752          MOV     @R0, A
040C A3          +1 753          INC     DPTR
040D 08          +1 754          INC     R0
040E DFFA          +1 755          DJNZ   R7, IVLoop2
0410 8849          +1 756          MOV     SerialIN, R0
0412 754A5F       +1 757          MOV     USBout, #ReceiveBuffer
0415 75484B       +1 758          MOV     SerialOUT, #SendBuffer
0418 75474B       +1 759          MOV     USBin, #SendBuffer
          +1 760
          +1 761
041B          +1 762      InitializeIOSystem:          ; Work around the Dscope monitor I/O needs
          +1 763      ; Use Serial 0 since dScope has Serial 1
041B 74FF          +1 764          MOV     A, #0FFH        ; Set up Timer 2 as a baud rate generator

```

```

041D F5CD      +1  765          MOV     T2High, A
041F F5CC      +1  766          MOV     T2Low, A
0421 F5CB      +1  767          MOV     T2ReloadHigh, A
0423 74D9      +1  768          MOV     A, #0D9H           ; 19.2Kb
0425 F5CA      +1  769          MOV     T2ReloadLow, A
0427 7434      +1  770          MOV     A, #00110100b
0429 F5C8      +1  771          MOV     T2Control, A      ; Start Timer 2
042B 7450      +1  772          MOV     A, #01010000b    ; Set up Serial 0 as 1 start, 8 data, 1
stop
042D F598      +1  773          MOV     S0Control, A
          +1  774
042F          +1  775          InitializeInterruptSystem: ; First initialize the USB level
042F 7400      +1  776          MOV     A, #00000000b
0431 78AC      +1  777          MOV     R0, #LOW(IN07IEN)
0433 F2        +1  778          MOVX   @R0, A           ; Enable EP1 IN Endpoints only
0434 E4        +1  779          CLR    A
0435 08        +1  780          INC    R0
0436 F2        +1  781          MOVX   @R0, A           ; Disable interrupts from OUT Endpoints 0-7
0437 08        +1  782          INC    R0
0438 7403      +1  783          MOV     A, #00000011b
043A F2        +1  784          MOVX   @R0, A           ; Enable (Resume, Suspend,) SOF and SUDAV
INTs
043B 08        +1  785          INC    R0
043C 7401      +1  786          MOV     A, #00000001b
043E F2        +1  787          MOVX   @R0, A           ; Enable Auto Vectoring for USB interrupts
          +1  788
043F 78B6      +1  789          MOV     R0, #LOW(EP1Control) ; Clear any stale information from ...
0441 7402      +1  790          MOV     A, #00000010b      ; ... EP1 IN buffer
0443 F2        +1  791          MOVX   @R0, A
          +1  792          ; Now enable the main level
0444 75E801    +1  793          MOV     EIE, #00000001b    ; Enable INT2 = USB Interrupt (only)
0447 75A8D0    +1  794          MOV     EI, #11010000b    ; Enable Serial0 interrupt (and Ser1 for
dScope)
          +1  795
          +1  796          ; Initialization Complete.
          +1  797          ;
044A          +1  798          MAIN:
044A 00        +1  799          NOP                                ; Not much of a main loop for this example
044B 80FD      +1  800          JMP     MAIN                       ; All actions are initiated by interrupts
          +1  801          ; We are a slave, we wait to be told what to do
          +1  802
044D          +1  803          Wait100msec:
044D 754064    +1  804          MOV     Temp, #100
0450          +1  805          Wait1msec:                         ; A delay loop
0450 90FB50    +1  806          MOV     DPTR, #-1200
0453 A3        +1  807          More:  INC    DPTR                 ; 3 cycles
0454 E582      +1  808          MOV     A, DPL                   ; + 2
0456 4583      +1  809          ORL    A, DPH                     ; + 2
0458 70F9      +1  810          JNZ    More                       ; + 3 = 10 cycles x 1200 = 1msec
045A D540F3    +1  811          DJNZ   Temp, Wait1msec
045D 22        +1  812          RET
          +1  813
          +1  814
          +1  815
          +1  816          ;$include (Timer.A51)
          +1  817          ; This module services the real time interrupt
          +1  818          ; Not used in this example
045E          +1  819          ServiceTimerRoutine:
045E 22        +1  820          Done:  RET
          +1  821
          +1  822
          +1  823
          +1  824          END

```