```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST P=18F8722,F=INHX32,b=8,n=51,t=ON ;directive to define processor and file format
                     00002 ;*****************************************************************************
                     00003 ;   This file is the main relocatable assembly source code module for the
                     00004 ;   1401 Tape Adapter PIC18F8722.
                     00005 ;
                     00006 ;   The PIC18FXXXX architecture allows two interrupt configurations.
                     00007 ;       This code is written for priority interrupt levels and the IPEN bit
                     00008 ;   in the RCON register is set to enable priority levels.
                     00009 ;
                     00010 ;*****************************************************************************
                     00011 ;
                     00012 ;    Filename:  Main.asm
                     00013 ;    Date:      8/7/06
                     00014 ;    File Version: 1.0
                     00015 ;
                     00016 ;    Author:    Bob Feretich
                     00017 ;
                     00018 ;*****************************************************************************
                     00019 ;
                     00020 ;    Files required:    P18F8722.INC
                     00021 ;                       Main.INC
                     00022 ;                       Main.LKR
                     00023 ;
                     00024 ;       Naming conventions:
                     00025 ;       ACS fields              All uppercase, use _ to separate words
                     00026 ;       Other RAM fields        First letter lowercase, use cap to separate words
                     00027 ;       Flash fields/labels     First letter capatalized, use cap to separate words
                     00028 ;       Constant values for above fields:
                     00029 ;       <ROOT>_<constName>
                     00030 ;               where <ROOT> is built from the name of the field and has same capitalization rules.
                     00031 ;               where <constName> has first letter uppercase and caps separate words
                     00032 ;       Constants/Literals that are not associated with data fields    All uppercase, use _ to separate
                     00033 ;       My assembler variables  All lowercase and uses _ to separate words
                     00034 ;
                     00035 ;       Loop labeling convention:
                     00036 ;       <subroutine_name>L<loop_seqnum>s = start if loop
                     00037 ;       <subroutine_name>L<loop_seqnum>c = bottom, but inside loop, similar to CONTINUE
                     00038 ;       <subroutine_name>L<loop_seqnum>x = exit loop, similar to BREAK or LEAVE
                     00039 ;       <subroutine_name>L<loop_seqnum>s = start if loop
                     00040 ;
                     00041 ;      Endian conventions:
                     00042 ;       When data is stored in program memory of PIC18 devices, the first character
                     00043 ;       (lowest memory address) is in least significant byte.
                     00044 ;       The lowest order bit in a field is bit 0.
                     00045 ;
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00046 ;      Stack use convention:
                     00047 ;       Stack grows from lower to higher addresses. So "top of stack" is the highest
                     00048 ;         address used.
                     00049 ;       Push operstions pre-increment the stack pointer, then move data the "top of
                     00050 ;         stack" location.
                     00051 ;       Pop operations post-decriment the stack pointer.
                     00052 ;       PLUSW accesses therefore must index the stack with negative number offsets.
                     00053 ;       Extended mode is not enabled, so indexed access to the stacks are limited.
                     00054 ;       FSR0 is the software stack used by all applications and interrupt levels.
                     00055 ;       FSR1 is reserved for high level interrupt handler use. Contents are not saved
                     00056 ;         at interrupt entry. The contents are not expected to be valid at interrupt entry.
                     00057 ;         Another interrupt handler at the same level may have changed its contents.
                     00058 ;       FSR2 is available for use by all code. FSR2 must be saved and restored by
                     00059 ;         interrupt handlers.
                     00060 ;
                     00061 ;      Multibyte precision convention:
                     00062 ;       In storage, lowest order byte in lowest storage address of consecutive locations.
                     00063 ;       In REGs, lowest order byte in lowest numbered register of consecutive registers.
                     00064 ;
                     00065 ;      Reset definition:
                     00066 ;       1) Min Reset (MR) - This is the reset that occurs if the Adapter detects an
                     00067 ;          internal error. The Box Reset resets the minimum amount of Adapter facilities
                     00068 ;          needed to continue insrtuction execution. The amount of retained state is
                     00069 ;          maximized to aid in the diagnosis of the error. RAM locations and groups of
                     00070 ;          RAM locations that are reinitialized upon this reset are marked with the
                     00071 ;          letters IMR.
                     00072 ;       2) Software Reset (SR) - This is the reset that is performed by the "Full Reset"
                     00073 ;          Command Message. A Software Reset will reinitialize the Adapter application to
                     00074 ;          its power-on state so that it will be ready to accept USB Command messages.
                     00075 ;          All virtual drive and adapter state information will be reinitialized.
                     00076 ;          RAM locations and groups of RAM locations that are reinitialized upon this
                     00077 ;          reset are marked with the letters ISR.
                     00078 ;       3) Power-On-Reset (PR) - This is the most comprehensive reset. PIC timers and
                     00079 ;          I/O Ports are reinialized. All RAM locations are initialized. This reset is a
                     00080 ;          superset of Min Reset and Software Reset. RAM locations that do not have an
                     00081 ;          architected initial value are set to a filler character (0xAA). RAM locations
                     00082 ;          and groups of RAM locations that have an architected initial state, but are
                     00083 ;          not reset by MR or SR are marked with the letters IPR.
                     00084 ;
                     00085 ;       Hang codes:
                     00086 ;       0x01 = invalid usb state machine state
                     00087 ;       0x02 = false low priority interrupt
                     00088 ;       0x02 = false high priority interrupt
                     00089 ;
                     00090 ;*****************************************************************************
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00091 ;
                     00092         radix   dec
  00000001           00093 F       EQU     1               ; used in instructions to indicate "put result in RAM file".
  0000               00094         variable is_simulating  ; B'0'= compile for PIC execution
                     00095                                 ; B'1'= compile for simulation (no timimg loops)
  00000000           00096 is_simulating   set     B'0'
                     00097 ;
                     00098 ;*****************************************************************************
                     00099         #include <P18F8722.INC>         ;processor specific variable definitions
                     00001         LIST
                     00002
                     00003 ;==========================================================================
                     00004 ;  $Id: P18F8722.INC,v 1.2.4.2 2005/02/15 17:35:36 curtiss Exp $
                     00005 ;  MPASM PIC18F8722 processor include
                     00006 ;
                     00007 ;  (c) Copyright 1999-2005 Microchip Technology, All rights reserved
                     00008 ;==========================================================================
                     00009
                     02266         LIST
                     00100         #include Main.INC                      ;project macro file
                     00001 ;Main includes
                     00002 ;       Port Reg = value/8  Bit = value MOD 8
                     00003 ;       All names of the form PIN_<port><bit>
                     00004
                     00005 #define PIN_A0  31744
                     00006 #define PIN_A1  31745
                     00007 #define PIN_A2  31746
                     00008 ;   ...
                     00081         list
                     00082 #define PIN_J7  31815
                     00083
                     00084 output_high     macro   pin
                     00085                 bsf     (pin)>>3&0xff,(pin)&0x07,0
                     00086                 endm
                     00087
                     00088 output_low      macro   pin
                     00089                 bcf     (pin)>>3&0xff,(pin)&0x07,0
                     00090                 endm
                     00091
                     00092 input           macro   pin
                     00093                 bcf     STATUS,C
                     00094                 btfsc   (pin)>>3&0xff,(pin)&0x07,0
                     00095                 bsf     STATUS,C
                     00096                 endm
                     00097
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00098 delay_ms        macro   time
                     00099                 movlw   time
                     00100                 call    Delay_ms
                     00101                 endm
                     00101         TITLE   "1401 Tape Channel Analyzer Adapter,  Rev 0.1"
                     00102         SUBTITLE "Configuration setup section"
                     00103 ;*****************************************************************************
                     00104 ;Configuration bits
                     00105 ; The CONFIG directive defines configuration data within the .ASM file.
                     00106 ; The labels following the directive are defined in the P18F8722.INC file.
                     00107 ; The PIC18FXX22 Data Sheet explains the functions of the configuration bits.
                     00108 ; Change the following lines to suit your application.
                     00109
                     00110         CONFIG  OSC=HSPLL,FCMEN=OFF,IESO=OFF ;fail safe mon & switch disabled
                     00111         CONFIG  BOREN=OFF,PWRT=OFF      ;brown-out reset and power-up-timer disabled
                     00112         CONFIG  WDT=OFF                 ;watch-dog-timer disabled
                     00113         CONFIG  MODE=MC                 ;use microcontroller memory mode
                     00114         CONFIG  STVREN=ON,LVP=OFF,DEBUG=OFF ;stack-ovflw-reset, low-voltage ICSP, bkgnd-debug enabled
                     00115         CONFIG  CP0=OFF,CP1=OFF,CP2=OFF,CP3=OFF,CP4=OFF,CP5=OFF,CP6=OFF,CP7=OFF ;code protection off
                     00116         CONFIG  CPB=OFF,CPD=OFF         ;boot-block and eeprom protection disabled
                     00117         CONFIG  WRT0=OFF,WRT1=OFF,WRT2=OFF,WRT3=OFF,WRT4=OFF,WRT5=OFF,WRT6=OFF,WRT7=OFF ;write protectio
                     00118         CONFIG  WRTC=ON,WRTB=OFF,WRTD=OFF ;config, boot-block, & data-eeprom protection disabled
                     00119         CONFIG  EBTR0=OFF,EBTR1=OFF,EBTR2=OFF,EBTR3=OFF,EBTR4=OFF,EBTR5=OFF,EBTR6=OFF,EBTR7=OFF ;table r
                     00120         CONFIG  EBTRB=OFF               ;boot-block table read protection disabled
                     00121 ;*****************************************************************************
                     00122         radix   dec
                     00123 #define USB_WR  PIN_G2
                     00124 #define USB_RD  PIN_G1
                     00125 #define SND     PIN_G4
                     00126 #define TXE     PIN_B4
                     00127 #define RXF     PIN_B2
                     00128 #define LED     PIN_J5
  00000090           00129 TRIS_OFFSET     equ  144        ;offset from port's data reg to TRIS reg
                     00130
                     00131 ;       Port D selection decode applied to Port E bits 2-0
  00000000           00132 SELUSB          EQU     0       ;FT245 FIFO
  00000001           00133 SELREGOE        EQU     1       ;Select Register output enable
  00000002           00134 SELRCDR         EQU     2       ;Read Channel Polarity (Direction) Register
  00000003           00135 SELRCPR         EQU     3       ;Read Channel Pulse Register
  00000004           00136 SELRCAA         EQU     4       ;Read Channel Amplitude A Register
  00000005           00137 SELRCAB         EQU     5       ;Read Channel Amplitude B Register
  00000006           00138 SELRCAC         EQU     6       ;Read Channel Amplitude C Register
  00000007           00139 SELRCAD         EQU     7       ;Read Channel Amplitude D Register
                     00140
```

LOC OBJECT CODE   LINE SOURCE TEXT
  VALUE

```
                     00141  PAGE
                     00142                  SUBTITLE        "ACCESS DATA MEMORY definition section"
                     00143                  UDATA_ACS
                     00144
                     00145 ;*****************************************************************************
                     00146 ;*****************************************************************************
                     00147 ;*******  ACS locations are all CAPS Literals are also all Caps
                     00148 ;*****************************************************************************
                     00149 ;*****************************************************************************
                     00150 ;       High Priority Interrupt Level ACS memory        8 bytes
                     00151 ;*****************************************************************************
000000               00152 HPSVCCNT        RES 1           ;Count of interrupts handled in pass
                     00153 ;       working registers
000001               00154 HPREG0          RES 1
000002               00155 HPREG1          RES 1
000003               00156 HPREG2          RES 1
000004               00157 HPREG3          RES 1
000005               00158 HPREG4          RES 1
000006               00159 HPREG5          RES 1
000007               00160 HPREG6          RES 1
000008               00161 HPREG7          RES 1
                     00162
                     00163 ;*****************************************************************************
                     00164 ;       Low Priority Interrupt Level ACS memory         18 bytes
                     00165 ;*****************************************************************************
000009               00166 LPSVCCNT        RES 1           ;Count of interrupts handled in pass
                     00167 ;       Working registers
00000A               00168 LPREG0          RES 1
00000B               00169 LPREG1          RES 1
00000C               00170 LPREG2          RES 1
00000D               00171 LPREG3          RES 1
00000E               00172 LPREG4          RES 1
00000F               00173 LPREG5          RES 1
000010               00174 LPREG6          RES 1
000011               00175 LPREG7          RES 1
                     00176
                     00177 ;       Tape Channel Signal data port snapshot
000012               00178 PDAT_TS         RES 4   ;Timestamp for sample
000016               00179 PDAT_A          RES 1   ;Port A
000017               00180 PDAT_B          RES 1   ;Port B
000018               00181 PDAT_C          RES 1   ;Port C
000019               00182 PDAT_E          RES 1   ;Port E
00001A               00183 PDAT_H          RES 1   ;Port H
                     00184
                     00185 ;       Free Running Timer RAM Extension (ISR)
```

ACCESS DATA MEMORY definition section
LOC OBJECT CODE   LINE SOURCE TEXT
  VALUE

```
00001B                00186 FRTIMER_B1      RES 1   ;Bits 23-16 of the free running timer
00001C                00187 FRTIMER_B2      RES 1   ;Bits 31-24 of the free running timer
00001D                00188 FRTIMER_B3      RES 1   ;Bits 39-32 of the free running timer
                      00189
                      00190 ;*****************************************************************************
                      00191 ;*****************************************************************************
                      00192 ;       Tape Adapter State (part 1, access ram part)            34 bytes
                      00193 ;*****************************************************************************
                      00194 ;*****************************************************************************
00001E                00195 ANLS_STATUS     RES 1   ;all zeroes is good (ISR)
  00000007            00196 ANLS_MisXfer    EQU  H'0007'    ;host did not respond in time
  00000006            00197 ANLS_BufOflw    EQU  H'0006'    ;buffer overflowed (wrap attempted), data was discarded
  00000005            00198 ANLS_BufUflw    EQU  H'0005'    ;buffer underflowed, data demanded while buffer empty
  00000004            00199 ANLS_UsbFlag    EQU  H'0004'    ;USB received characters when expecting start msg flag
  00000003            00200 ANLS_UsbTimeout EQU  H'0003'    ;USB transmission has timed out, PC did not receive data
00001F                00201 ANLS_STATE      RES 1   ;state machine state (ISR)
000020                00202 ANLS_STATE2     RES 1   ;modifier to state machine state (ISR)
  00000001            00203 ANLS_FirstChar  EQU  H'0001'    ;character received on 1401 write bus
000021                00204 ANLS_RESR       RES 1   ;last reset reason code (RCON Register) (IPR)
                      00205 ;       Emulation and monitoring state
000022                00206 ANLS_MMASK      RES 1   ;bit mask of drives being monitored (ISR)
  00000006            00207 DRIVE6          EQU  H'0006'    ;bit for tape drive
  00000005            00208 DRIVE5          EQU  H'0005'    ;bit for tape drive
  00000004            00209 DRIVE4          EQU  H'0004'    ;bit for tape drive
  00000003            00210 DRIVE3          EQU  H'0003'    ;bit for tape drive
  00000002            00211 DRIVE2          EQU  H'0002'    ;bit for tape drive
  00000001            00212 DRIVE1          EQU  H'0001'    ;bit for tape drive
000023                00213 ANLS_EMASK      RES 1   ;bit mask of drives being emulated (ISR)
000024                00214 ANLS_ACTIVE_DRV RES 1   ;bit mask with the active drive's bit on (ISR)
                      00215                         ; we are "in session" with this drive
000025                00216 ANLS_DRVCNTXT   RES 2   ;pointer to the active drive's state area  (ISR)
                      00217 ;       buffer status
000027                00218 ANLS_BSTATUS    RES 1   ;(ISR)
  00000007            00219 ANLS_TDActive   EQU  H'0007'    ;1=usb port is transfering data to tape data buffer
                      00220 ;       tape data buffer pointers, buffer is empty when HEAD==TAIL
000028                00221 ANLS_TDHEAD     RES 2   ;location of most recent byte added  (ISR)
00002A                00222 ANLS_TDTAIL     RES 2   ;location of most recent byte removed  (ISR)
                      00223 ;
                      00224 ;       analyzer configuration
00002C                00225 ANLS_CNFG1      RES 1   ;flags, valid only for emulated drives (ISR)
  00000007            00226 ANLS_FrcNotRdy  EQU  H'0007'    ;drives are forced "NOT READY"
  00000006            00227 ANLS_FrcNoEP    EQU  H'0006'    ;drives will not generate "ECHO PULSES"
  00000005            00228 ANLS_FrcLRCE    EQU  H'0005'    ;force response with bad LRC character
  00000004            00229 ANLS_FrcVRCE    EQU  H'0004'    ;force response with bad VRC (parity)
  00000003            00230 ANLS_FrcEchoE   EQU  H'0003'    ;force echoed data error
```

1401 Tape Channel Analyzer Adapter, Rev 0.1
ACCESS DATA MEMORY definition section
LOC OBJECT CODE   LINE SOURCE TEXT
  VALUE

  00000002   00231 ANLS_GenLRC   EQU H'0002'   ;1=generate the LRC character (not in data)
  00000001   00232 ANLS_Model5or6 EQU H'0001'   ;1=respond with MOD_5_OR_6 active
  00000000   00233 ANLS_Model4   EQU H'0000'   ;1=respond using SEL&RDY_MOD4, 0=respond _MOD2
                      00234   ;using the encode of above two bits
                      00235   ;xxxx xx00 - means emulate an IBM 729 Model 2
                      00236   ;xxxx xx01 - means emulate an IBM 729 Model 4
                      00237   ;xxxx xx10 - means emulate an IBM 729 Model 5
                      00238   ;xxxx xx11 - means emulate an IBM 729 Model 6
                      00239 ;the below locations are used to set Tape Channel Read Bus Track Gain Controls
00002D   00240 ANLS_NGAIN21   RES 1   ;normal gain for these two tape channel tracks (ISR)
00002E   00241 ANLS_NGAIN84   RES 1   ;normal gain for these two tape channel tracks (ISR)
00002F   00242 ANLS_NGAINBA   RES 1   ;normal gain for these two tape channel tracks (ISR)
000030   00243 ANLS_NGAIN_C   RES 1   ;normal gain for this tape channel track (ISR)
000031   00244 ANLS_TGAIN21   RES 1   ;test gain for these two tape channel tracks
000032   00245 ANLS_TGAIN84   RES 1   ;test gain for these two tape channel tracks
000033   00246 ANLS_TGAINBA   RES 1   ;test gain for these two tape channel tracks
000034   00247 ANLS_TGAIN_C   RES 1   ;test gain for this tape channel track
                      00248 ;the below locations are for data generator control
000035   00249 ANLS_DGMODE   RES 1   ;data generator mode   (ISR)
  00000001   00250 ANLS_DGM_RP   EQU 1   ;repeat test character without alteration
  00000002   00251 ANLS_DGM_AL   EQU 2   ;alternate between test and filler characters
  00000003   00252 ANLS_DGM_CT   EQU 3   ;test by count, same as 0 except count is used
  00000004   00253 ANLS_DGM_BI   EQU 4   ;use BCD collating sequence to determine next char
  00000005   00254 ANLS_DGM_BR   EQU 5   ;binary rotate to form next test character
000036   00255 ANLS_DGTDLY   RES 2   ;test delay
000038   00256 ANLS_DGTDUR   RES 2   ;test duration
00003A   00257 ANLS_DGTRPT   RES 2   ;test repeat count
00003C   00258 ANLS_DGRECL   RES 2   ;generated record length
00003E   00259 ANLS_DGTCHR   RES 1   ;initial test data character
00003F   00260 ANLS_DGFCHR   RES 1   ;initial filler data character
000040   00261 ANLS_DGWCHR   RES 1   ;working data character
                      00262 ;
                      00263 ;   Semaphore for the Tape Data Buffer (ISR)
                      00264 ;   Semaphore change sequence for all threads is specified below:
                      00265 ;   Idle State: (TDS_SREQ==0 && TDS_RCODE==0) No activity in progress
                      00266 ;   Access privileges: TISM   Background
                      00267 ;   TDS_SREQ   R/W   RO (Read Only)
                      00268 ;   TDS_RCODE   RO   RO
                      00269 ;   Tape Buffer   R/W   None
                      00270 ;   Action - TISM can make a reguest by storing it into TDS_SREQ.
                      00271 ;   Request State: (TDS_SREQ!=0 && TDS_RCODE==0) Service requested.
                      00272 ;   Access privileges: TISM   Background
                      00273 ;   TDS_SREQ   RO   RO
                      00274 ;   TDS_RCODE   RO   R/W
                      00275 ;   Tape Buffer   None   R/W

LOC OBJECT CODE   LINE SOURCE TEXT
  VALUE

```
                     00276 ;               Action - Background thread can mark a request completed by storing
                     00277 ;                        a return code in TDS_RCODE.
                     00278 ;        Completed State: (TDS_SREQ!=0 && TDS_RCODE!=0)
                     00279 ;                                        Service complete but not acknowledged.
                     00280 ;               Access privileges:       TISM    Background
                     00281 ;                        TDS_SREQ        R/W     RO
                     00282 ;                        TDS_RCODE       RO      RO
                     00283 ;                        Tape Buffer     R/W     None
                     00284 ;               Action - TSIM thread can acknowledge completion by clearing TDS_SREQ.
                     00285 ;                        No action is necessary by the Background task.
                     00286 ;        Acknowledged State: (TDS_SREQ0=0 && TDS_RCODE!=0) Service acknowledged.
                     00287 ;               Access privileges:       TISM    Background
                     00288 ;                        TDS_SREQ        RO      RO
                     00289 ;                        TDS_RCODE       R/W     RO
                     00290 ;                        Tape Buffer     R/W     None
                     00291 ;               Action - TSIM thread can cause transition to Idle State by clearing
                     00292 ;                        TDS_RCODE. No action is necessary by the Background task.
                     00293 ;       Note that the TISM thread can transition the semaphore from Completed State
                     00294 ;       to Acknowledged State, then immediately into Idle State, then immediately to
                     00295 ;       Request State without the Background thread's participation.
                     00296 ;
                     00297 ;       The Background thread shall detect service requests by:
                     00298 ;               1) Check that TDS_RCODE==0, if it's not, the TDS_SREQ field may
                     00299 ;                        contain an old service request.
                     00300 ;               2) Check that TDS_SREQ!=0, if it is zero, then no service is being
                     00301 ;                        requested.
                     00302 ;
000041               00303 TDS_SREQ        RES 1   ;0=no service requested, access to TD Buffer prohibited
                     00304                         ;!0=service is requested, access to TD Buffer granted
                     00305                         ;The value indicates the service requested.
                     00306                         ;Valid requests are:
                     00307                         ; usbEv_DnldReq = request download of data from the pc
                     00308                         ; usbEv_UpldReq = reguest upload of data to the pc
                     00309                         ;This byte is written only by the Tape Interface State
                     00310 ;                       ;Machine thread.
000042               00311 TDS_RCODE       RES 1   ;Tape data buffer service return code.
                     00312                         ;Written (posted) by the Background Thread.
                     00313                         ;Cleared by the Tape Interface State Machine thread.
                     00314                         ;Valid postings are: tbd
                     00315
                     00316 ;       Semaphore for the Event Data Buffer (ISR)
                     00317 ;       The semaphore change sequence and service detection sequence for this
                     00318 ;       semaphore is the same as for the Tape Data Buffer Semaphore. Access
                     00319 ;       privileges for EDS_COUNT is the same as for the Event Buffer.
000043               00320 EDS_SREQ        RES 1   ;0=no service requested, access to Event Buffer prohibited
```

LOC OBJECT CODE   LINE SOURCE TEXT
  VALUE

```
                    00321                         ;!0=service is requested, access to Event Buffer granted
                    00322                         ;The value indicates the service requested.
                    00323                         ;Valid requests are:
                    00324                         ; usbEv_DnldReq = request download of data from the pc
                    00325                         ; usbEv_DnldUndflw = requested download took to long
                    00326                         ; usbEv_UpldReq = reguest upload of data to the pc
                    00327                         ; usbEv_UpldOvrflw = requested upload took too long
                    00328                         ; usbEv_Backspace = notification of backspace operation
                    00329                         ; usbEv_Rewind = notification of rewind operation
                    00330                         ; usbEv_Unload = notification of rewind & unload operation
                    00331                         ; usbEv_StatusChg = notification of change in status
                    00332                         ; usbEv_RdTimeout = notification of timeout occurrence
                    00333                         ; usbEv_WrtTimeout = notification of timeout occurrence
                    00334                         ; usbEv_BksTimeout = notification of timeout occurrence
                    00335                         ; usbEv_ChanTrans = notification of tape channel activity
                    00336                         ;This byte is written only by the Tape Interface State
                    00337 ;                       ;Machine thread.
000044              00338 EDS_RCODE       RES 1   ;Tape data buffer service return code.
                    00339                         ;Written (posted) by the Background Thread.
                    00340                         ;Cleared by the Tape Interface State Machine thread.
                    00341                         ;Valid postings are: 1=complete
000045              00342 EDS_COUNT       RES 1   ;The number of bytes in the buffer.
                    00343
                    00344 ;*******************************************************************************
                    00345 ;       Background ACS memory           xx bytes
                    00346 ;*******************************************************************************
                    00347 ;       working registers               8 bytes
000046              00348 REG0            RES 1
000047              00349 REG1            RES 1
000048              00350 REG2            RES 1
000049              00351 REG3            RES 1
00004A              00352 REG4            RES 1
00004B              00353 REG5            RES 1
00004C              00354 REG6            RES 1
00004D              00355 REG7            RES 1
                    00356
```

LOC OBJECT CODE   LINE SOURCE TEXT
  VALUE

```
                     00357  PAGE
                     00358                  SUBTITLE "DATA MEMORY definition section"
                     00359 URAM    UDATA
                     00360
                     00361 ;*****************************************************************************
                     00362 ;       Tape Adapter State (part 2, non-access ram part)
                     00363 ;*****************************************************************************
                     00364 ;
                     00365 ;       Diagnostic command state data (from DiagUpload Loopback command)
000000               00366 diagCntl        res 1   ;Diagnostic instruction control mask (ISR)
  00000000           00367 diag_UpLdLb     equ     0       ;DiagUpload Loopback active bit
000001               00368 diagAlarmTime   res 4   ;Free running timer timestamp format (low byte first)
000005               00369 diagDelayTime   res 4   ;Free running timer timestamp format (low byte first)
000009               00370 diagRepeatCnt   res 2   ;Total repatitions of the event (0=send only one)
                     00371
                     00372 ;*****************************************************************************
                     00373 ;       Tape Drive State        8 bytes per drive x 6 drives = 48 bytes
                     00374 ;*****************************************************************************
                     00375 ;       Drive status template
  00000000           00376 DSTATE          EQU 0   ;byte
  00000007           00377 ds_Selected     EQU  H'0007'    ;selected
  00000006           00378 ds_MechRdy      EQU  H'0006'    ;mechanically ready
  00000005           00379 ds_ElecRdy      EQU  H'0005'    ;electrically ready
  00000004           00380 ds_WRTMODE      EQU  H'0004'    ;1=write mode, 0=read mode
  00000003           00381 ds_HiDensity    EQU  H'0003'    ;1=high density, 0=low density mode
  00000002           00382 ds_TapeInd      EQU  H'0002'    ;tape indicator state
  00000001           00383 ds_WriteEnb     EQU  H'0001'    ;1=write enabled, 0=write disabled
  00000000           00384 ds_Bit0         EQU  H'0000'    ;unused
  00000001           00385 DS_LRC          EQU 1           ;byte   ;pending check character
  00000002           00386 DS_TAPEPOS      EQU 2           ;int    ;# of rectords from start of tape, 0=at load point
  00000004           00387 DS_TYPE         EQU 4           ;byte model configuration 2=ModII, 4=ModIV, 5=ModV
00000B               00388 drive1State     RES 8   ;state of tape drive 1
000013               00389 drive2State     RES 8   ;state of tape drive 2
00001B               00390 drive3State     RES 8   ;state of tape drive 3
000023               00391 drive4State     RES 8   ;state of tape drive 4
00002B               00392 drive5State     RES 8   ;state of tape drive 5
000033               00393 drive6State     RES 8   ;state of tape drive 6
```

DATA MEMORY definition section
LOC OBJECT CODE   LINE SOURCE TEXT
  VALUE

```
                      00394  PAGE
                      00395 ;*******************************************************************************
                      00396                SUBTITLE        "USB control locations and definitions"
                      00397 ;*******************************************************************************
                      00398 ; USB Interface State Machine           9 bytes
                      00399 ;*******************************************************************************
                      00400 ; USB Command Envelope
                      00401 ;       Start Flag - 1 byte 0xf0
                      00402 ;       Message length - 2 bytes (length of command specific data + 2)
                      00403 ;       Command code - 1 byte
                      00404 ;       Command specific data - variable length
                      00405 ;       CRC - 1 byte (makes the unsigned sum of all non-flag bytes modulo 256 equal zero)
                      00406 ;       Stop Flag - 1 byte 0xf1
                      00407 ; USB Command Response and Event Envelope
                      00408 ;       Start Flag - 1 byte 0xf0
                      00409 ;       Message length - 2 bytes (length of command response data + 6)
                      00410 ;       Command code - 1 byte (the code of the Command that caused this response.)
                      00411 ;       time_stamp - 4 bytes
                      00412 ;       Command response data - variable length
                      00413 ;       CRC - 1 byte (makes the unsigned sum of all non-flag bytes modulo 256 equal zero)
                      00414 ;       Stop Flag - 1 byte 0xf1USB_ACCUM_CRC    RES 1   ;accumulated CRC
00003B                00415 usbState        RES 1   ;Parser is expecting a specific byte in the message packet (ISR)
  00000000            00416 usbSt_ExpBegin  equ 0x00        ;exoecting the begin flag
  00000001            00417 usbSt_ExpLen1   equ 0x01        ;expecting the first byte of msg length (low order)
  00000002            00418 usbSt_ExpLen2   equ 0x02        ;expecting the second byte of the length (high order)
  00000003            00419 usbSt_ExpCmd    equ 0x03        ;expecting the command code
  00000004            00420 usbSt_ExpData   equ 0x04        ;expecting data until USB count = 1
  00000005            00421 usbSt_ExpCRC    equ 0x05        ;expecting CRC byte
  00000006            00422 usbSt_ExpStop   equ 0x06        ;expecting the stop flag
  00000007            00423 usbSt_ExpPurge  equ 0x07        ;expecting trash characters, they are ignored
00003C                00424 usbStatus       RES 1   ;status buts for msg handling, acted upon during msg response (ISR)
  00000001            00425 usbSs_Trunc     equ 0x01        ;received message was truncated
  00000002            00426 usbSs_BadCRC    equ 0x02        ;crc error
  00000003            00427 usbSs_InvCmd    equ 0x03        ;invalid command code
  00000004            00428 usbSs_BufOflw   equ 0x04        ;receive buffer overflowed
  00000007            00429 usbSs_StTape    equ 0x07        ;1=store next data in tape buffer, 0=usb receive buffer
00003D                00430 usbLengthL      RES 1   ;low order byte of two byte length field
00003E                00431 usbLengthH      RES 1   ;high order byte of two byte lengthe field
00003F                00432 usbCountL       RES 1   ;the number of command specific characters received + 1 (low order)
000040                00433 usbCountH       RES 1   ;the number of command specific characters received + 1 (high order)
000041                00434 usbCommand      RES 1   ;USB message command code
000042                00435 usbCRC          RES 1   ;the crc on the received data
000043                00436 usbRecNxt       RES 2   ;address of the next available location in the USB receive buffer
000045                00437 usbTapeDataNxt  RES 2   ;address of the next available location in the Tape Data buffer
                      00438 ;
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00439 ;       The below fields are for simulation purposes only
000047               00440 usbSimNxt       RES 3   ;address of next faked usb message byte in program storage
                     00441
                     00442 ; Command/Response codes
  00000000           00443 usbCmd_NotRcvd   equ    0x00    ; length of message was zero
  00000001           00444 usbCmd_FullReset equ    0x01
  00000002           00445 usbCmd_GetAnlCfg equ    0x02
  00000003           00446 usbCmd_SetAnlCfg equ    0x03
  00000004           00447 usbCmd_DefVirtDrv equ   0x04
  00000005           00448 usbCmd_UdefVirtDrv  equ 0x05
  00000006           00449 usbCmd_GetVirtDrvSt equ 0x06
  00000007           00450 usbCmd_SetVirtDrvSt equ 0x07
  00000008           00451 usbCmd_UpldRecord equ   0x08
  00000009           00452 usbCmd_DnldRecord equ   0x09
  00000010           00453 usbCmd_DiagUpldLp equ   0x10
                     00454 ; Event codes
  00000081           00455 usbEv_DnldReq   equ     0x81
  00000082           00456 usbEv_DnldUndflw equ    0x82
  00000083           00457 usbEv_UpldReq   equ     0x83
  00000084           00458 usbEv_UpldOvrflw equ    0x84
  00000085           00459 usbEv_Backspace equ     0x85
  00000086           00460 usbEv_Rewind    equ     0x86
  00000087           00461 usbEv_Unload    equ     0x87
  00000088           00462 usbEv_StatusChg equ     0x88
  00000089           00463 usbEv_RdTimeout equ     0x89
  0000008A           00464 usbEv_WrtTimeout equ    0x8A
  0000008B           00465 usbEv_BksTimeout equ    0x8B
  0000008C           00466 usbEv_ChanTrans equ     0x8C
                     00467 ; Exception Events
  000000FC           00468 usbEv_InvalidCmd equ    0xFC
  000000FD           00469 usbEv_CRCError  equ     0xFD
  000000FE           00470 usbEv_TruncMsg  equ     0xFE
                     00471 ; Flag Values
  000000F0           00472 USB_START_FLAG  equ     0xf0
  000000F1           00473 USB_STOP_FLAG   equ     0xf1
                     00474 ; USB message return codes
  00000000           00475 RC_SUCCESS      EQU 0
  00000001           00476 RC_BAD_ADDR     EQU 1
                     00477
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00478  PAGE
                     00479          SUBTITLE "Non-banked Buffers"
                     00480 ;*******************************************************************************
                     00481 ;       USB Command/Response Message Buffer
                     00482 ;       Messages received from the USB port are placed in this buffer.
                     00483 ;       Responses to these messages are also constructed in this buffer.
                     00484 ;       This buffer os owned by the background task and not shared.
                     00485 ;       When Command messages or Command Response messages contain tape
                     00486 ;       "Read Data" or "Write Data", only the first part of the message is
                     00487 ;       placed in this buffer is not stored in this buffer. The tape data is
                     00488 ;       transmitted from or stored into the Tape Data Buffer.
                     00489 ;*******************************************************************************
  00000500           00490 uBuffer          EQU 0x0500      ;room for 128 bytes
  0000057F           00491 uBufferEnd       EQU 0x057f
                     00492
                     00493 ;*******************************************************************************
                     00494 ;       USB Event Data Buffer
                     00495 ;       The Tape Interface State Machine (TISM) thread constructs event
                     00496 ;       messages in this buffer. Once the event message is constructed, the
                     00497 ;       Background thread transmits the event message.
                     00498 ;       The Event Data Buffer Semaphore (EDS_) controls the sharing of this
                     00499 ;       buffer.
                     00500 ;*******************************************************************************
  00000580           00501 eBuffer          EQU 0x0580      ;room for 128 bytes
  000005FF           00502 eBufferEnd       EQU 0x05ff
                     00503
                     00504 ;*******************************************************************************
                     00505 ;       Tape Drive Data Buffer (in unbanked memory)
                     00506 ;       Tape Read and Write data is placed in this buffer.
                     00507 ;       Although the Tape Interface State Machine (TISM) thread owns this buffer,
                     00508 ;       it does not access data on the buffer. The TISM requests services from
                     00509 ;       the background thread to move data between the buffer and the USB port.
                     00510 ;       The TISM uses the High Priority thread to move data between the 1401
                     00511 ;       Tape Channel and the buffer.
                     00512 ;       The Tape Data Buffer Semaphore (TDS_) controls the sharing of this
                     00513 ;       buffer.
                     00514 ;*******************************************************************************
  00000600           00515 tBuffer          EQU 0x0600      ;room for 2048 characters of read/write data
  00000DFF           00516 tBufferEnd       EQU 0x0dff
                     00517
                     00518 ;*******************************************************************************
                     00519 ;       Stack
                     00520 ;       Used for saving facilities during subroutine execution
                     00521 ;       Starts at lowest address and grows toward highest
                     00522 ;*******************************************************************************
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

  00000E00            00523 stack           EQU 0x0e00      ;room for 256 entries
  00000EFF            00524 stackEnd        EQU 0x0eff
                      00525
                      00526 ;*****************************************************************************
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00527  PAGE
                     00528                 SUBTITLE "Interrupt vector definition section"
                     00529 ResetVector     CODE    0x0000
                     00530
000000 EF?? F???     00531                 goto    PwrOnReset ;go to power-on-reset segment
                     00532
                     00533 ;*****************************************************************************
                     00534 ;High priority interrupt vector
                     00535 ; This code will start executing when a high priority interrupt occurs or
                     00536 ; when any interrupt occurs if interrupt priorities are not enabled.
                     00537
                     00538 HighIntVector   CODE    0x0008
                     00539
000008 EF?? F???     00540                 goto    HighInt ;go to high priority interrupt routine
                     00541
                     00542 ;*****************************************************************************
                     00543 ;Low priority interrupt vector
                     00544 ; This code will start executing when a low priority interrupt occurs.
                     00545 ; This code can be removed if low priority interrupts are not used.
                     00546
                     00547 LowIntVector    CODE    0x0018
                     00548
000018 EF?? F???     00549                 goto    LowInt  ;go to low priority interrupt routine
                     00550
                     00551 ;*****************************************************************************
                     00552                 SUBTITLE "High prority interrupt handler section"
                     00553 ;High priority interrupt routine
                     00554 ; The high priority interrupt code is placed here.
                     00555
                     00556                 CODE
                     00557
000000               00558 HighInt:
                     00559
                     00560 ;       *** high priority interrupt code goes here ***
                     00561
                     00562
000000 0011          00563                 retfie  FAST
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00564  PAGE
                     00565 ;*******************************************************************************
                     00566                 SUBTITLE "Low prority interrupt handler section"
                     00567 ;*******************************************************************************
000002               00568 LowInt: ;Save the STATUS reg, WREG, BSR, and FSR2
000002 CFD8 FFEC     00569                 movff   STATUS,PREINC0
000006 CFE8 FFEC     00570                 movff   WREG,PREINC0
00000A CFE0 FFEC     00571                 movff   BSR,PREINC0
00000E CFD9 FFEC     00572                 movff   FSR2L,PREINC0
000012 CFDA FFEC     00573                 movff   FSR2H,PREINC0
                     00574 ;       Set interrupts handled count to zero
000016 6A??          00575                 clrf    LPSVCCNT
                     00576 ;       if the Free Running Timer (Timer1) interrupted
000018 A09E          00577                 btfss   PIR1,TMR1IF
00001A D???          00578                 bra     LowIntIf1x
                     00579 ;         Increment the number of interrupts handled
00001C 2A??          00580                 incf    LPSVCCNT
                     00581 ;         Call the Timer1 Interrupt Handler
00001E EC?? F???     00582                 call    Timer1Int
                     00583 ;       endif Free Running Timer (Timer1) interrupted
000022               00584 LowIntIf1x:
                     00585 ;       if the interrupts handled count == zero
000022 50??          00586                 movf    LPSVCCNT,W
000024 E1??          00587                 bnz     LowIntRtn
                     00588 ;         The PIC saw an interrupt, but no device is claiming that it needs
                     00589 ;         service, so something has gone very wrong.
                     00590                 ;hang   2
                     00591 ;       endif the interrupts handled count == zero
                     00592 ;Rtn    Restore the STATUS reg, WREG, BSR, and FSR2
000026 CFED FFDA     00593 LowIntRtn       movff   POSTDEC0,FSR2H
00002A CFED FFD9     00594                 movff   POSTDEC0,FSR2L
00002E CFED FFE0     00595                 movff   POSTDEC0,BSR
000032 CFED FFE8     00596                 movff   POSTDEC0,WREG
000036 CFED FFD8     00597                 movff   POSTDEC0,STATUS
00003A 0010          00598                 retfie
```

1401 Tape Channel Analyzer Adapter, Rev 0.1
Low prority interrupt handler section
LOC OBJECT CODE   LINE SOURCE TEXT
  VALUE

```
                     00599  PAGE
                     00600                 SUBTITLE "Power-On-Reset section"
                     00601 ;*****************************************************************************
                     00602 ;       Perform the Power-On-Reset
                     00603 ;       Initialize the PIC hardware.
                     00604 ;*****************************************************************************
00003C               00605 PwrOnReset:
                     00606 ;       Set up the software (save) stack
00003C EE0E F000     00607                 lfsr    FSR0,stack
                     00608 ;       Perform Software Reset
                     00609                 ;call   MinReset
                     00610 ;       Initialize all data port directions to inputs
000040 0EFF          00611                 movlw   0xff
000042 6E92          00612                 movwf   TRISA           ;SET_TRIS_A     all input bits
000044 6E93          00613                 movwf   TRISB           ;SET_TRIS_B     all input bits
000046 6E94          00614                 movwf   TRISC           ;SET_TRIS_C     all input bits
000048 6E95          00615                 movwf   TRISD           ;SET_TRIS_D     all input bits
00004A 6E96          00616                 movwf   TRISE           ;SET_TRIS_E     all input bits
00004C 6E97          00617                 movwf   TRISF           ;SET_TRIS_F     all input bits
00004E 6E98          00618                 movwf   TRISG           ;SET_TRIS_G     all input bits
000050 6E99          00619                 movwf   TRISH           ;SET_TRIS_H     all input bits
000052 6E9A          00620                 movwf   TRISJ           ;SET_TRIS_J     all input bits
                     00621 ;       Default to all digital I/O
000054 0E0F          00622                 movlw   0x0f            ;setup_adc_ports( NO_ANALOGS )
000056 6EC1          00623                 movwf   ADCON1          ;
                     00624 ;       Disable the A/D converter
000058 6AE8          00625                 clrf    WREG            ;setup_adc(ADC_OFF )
00005A 6EC2          00626                 movwf   ADCON0          ;
                     00627 ;*****************************************************************************
                     00628 ;       Write the filler pattern throughout all of RAM
                     00629 ;*****************************************************************************
00005C EE2F F0FF     00630                 lfsr    FSR2,4095
000060 0EAA          00631                 movlw   0xAA
000062 66D9          00632 PORL1b          tstfsz  FSR2L
000064 D???          00633                 bra     PORL1c
000066 66DA          00634                 tstfsz  FSR2H
000068 D???          00635                 bra     PORL1c
00006A D???          00636                 bra     PORL1x
00006C 6EDD          00637 PORL1c          movwf   POSTDEC2
00006E D???          00638                 bra     PORL1b
000070               00639 PORL1x:
                     00640 ;*****************************************************************************
                     00641 ;       Initialize USB FIFO controls, then set pins to output mode
                     00642 ;*****************************************************************************
                     00643                 output_high     USB_WR                  ;init USB WR
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

000070 8486              M                bsf     (31794)>>3&0xff,(31794)&0x07,0
                      00644                output_high     USB_RD                  ;init USB RD
000072 8286              M                bsf     (31793)>>3&0xff,(31793)&0x07,0
                      00645                output_high     SND                     ;init SND
000074 8886              M                bsf     (31796)>>3&0xff,(31796)&0x07,0
                      00646                output_low      LED                     ;turn on LED
000076 9A88              M                bcf     (31813)>>3&0xff,(31813)&0x07,0
                      00647                output_low      USB_WR+TRIS_OFFSET      ;set TRIS to output
000078 9498              M                bcf     (31794+TRIS_OFFSET)>>3&0xff,(31794+TRIS_OFFSET)&0x07,0
                      00648                output_low      USB_RD+TRIS_OFFSET      ;set TRIS to output
00007A 9298              M                bcf     (31793+TRIS_OFFSET)>>3&0xff,(31793+TRIS_OFFSET)&0x07,0
                      00649                output_low      SND+TRIS_OFFSET         ;set TRIS to output
00007C 9898              M                bcf     (31796+TRIS_OFFSET)>>3&0xff,(31796+TRIS_OFFSET)&0x07,0
                      00650                output_low      LED+TRIS_OFFSET         ;set TRIS to output
00007E 9A9A              M                bcf     (31813+TRIS_OFFSET)>>3&0xff,(31813+TRIS_OFFSET)&0x07,0
                      00651 ;*****************************************************************************
                      00652 ;       Initialize the Adapter's Main and Analog Boards
                      00653 ;*****************************************************************************
                      00654 ;       tbd
                      00655 ;*****************************************************************************
                      00656 ;       Perform Software Reset
                      00657 ;*****************************************************************************
                      00658                ;call   SoftwareReset
                      00659 ;*****************************************************************************
                      00660 ;       Purge USB FIFO
                      00661 ;*****************************************************************************
                      00662        if (!is_simulating)
                      00663                delay_ms        100     ;time for the USB circuit to settle
000080 0E64              M                movlw   100
000082 EC?? F???         M                call    Delay_ms
                      00664        endif
                      00665 PWR_02:        input   RXF             ;while(!input(RXF)) {
000086 90D8              M                bcf     STATUS,C
000088 B481              M                btfsc   (31754)>>3&0xff,(31754)&0x07,0
00008A 80D8              M                bsf     STATUS,C
00008C E2??           00666               bc      PWR_01          ; (puts input bit in CARRY)
                      00667                output_low  USB_RD      ;
00008E 9286              M                bcf     (31793)>>3&0xff,(31793)&0x07,0
                      00668                output_high USB_RD      ;
000090 8286              M                bsf     (31793)>>3&0xff,(31793)&0x07,0
000092 D???           00669               bra     PWR_02
000094                00670 PWR_01:                                ;} //end while
                      00671 ;*****************************************************************************
                      00672 ;       Initialize Timers
                      00673 ;*****************************************************************************
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00674 ;       Initialize and start Timer1
                     00675 ;         Software reset already reset the RAM extension for the timer.
000094 EC?? F???     00676                 call    Timer1Init
                     00677 ;*******************************************************************************
                     00678 ;       Enable Interrupts
                     00679 ;       During this phase of testing all interrupts that are architected as
                     00680 ;       high priority will be configured as low priority interrupts.
                     00681 ;       This section will need to be rewritten in the final program.
                     00682 ;*******************************************************************************
                     00683 ;       Enable Interrupts
000098 0EC0          00684                 movlw   b'11000000'     ;GIEH+GIEL
Warning[202]: Argument out of range.  Least significant bits used.
00009A 6FF2          00685                 movwf   INTCON,GIE
                     00686 ;*******************************************************************************
                     00687 ;       Indicate to user that the Power-On-Reset has completed
                     00688 ;*******************************************************************************
                     00689 ;       flash a countdown 3, 2, 1
00009C 0E03          00690                 movlw   3
                     00691         if (!is_simulating)
00009E EC?? F???     00692                 call    FlashCntDn
                     00693         endif
                     00694
                     00695 ;       start main loop
0000A2 EF?? F???     00696                 goto    Main            ;goto main background loop
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00697  PAGE
                     00698 ;*******************************************************************************
                     00699                 SUBTITLE "Subroutines"
                     00700 ;       Flash LED for WREG times (1 minimum, 0 means 256 flashes)
                     00701 ;       Uses 6 stack locations (two here and four in delay_ms).
                     00702 ;       Returns with W maintained.
0000A6               00703 FlashLED:
0000A6 6EEC          00704                 movwf   PREINC0         ;push saved count value on stack
0000A8 6EEC          00705                 movwf   PREINC0         ;push working flash count value on stack
0000AA 4EEF          00706 Flash1          dcfsnz  INDF0           ;reduce flash counter
0000AC D???          00707                 bra     Flashx          ;exit if counter is zero
                     00708                 output_high LED         ;turn off the LED
0000AE 8A88              M                 bsf     (31813)>>3&0xff,(31813)&0x07,0
                     00709                 delay_ms    250         ;wait 250 mSec
0000B0 0EFA              M                 movlw   250
0000B2 EC?? F???         M                 call    Delay_ms
                     00710                 output_low  LED         ;turn on the LED
0000B6 9A88              M                 bcf     (31813)>>3&0xff,(31813)&0x07,0
                     00711                 delay_ms    250         ;wait 250 mSec
0000B8 0EFA              M                 movlw   250
0000BA EC?? F???         M                 call    Delay_ms
0000BE D???          00712                 bra     Flash1
0000C0               00713 Flashx          ;pause a second, then exit
                     00714                 output_high LED         ;turn off the LED
0000C0 8A88              M                 bsf     (31813)>>3&0xff,(31813)&0x07,0
                     00715                 delay_ms 250            ;wait 250 mSec
0000C2 0EFA              M                 movlw   250
0000C4 EC?? F???         M                 call    Delay_ms
                     00716                 delay_ms 250            ;wait 250 mSec
0000C8 0EFA              M                 movlw   250
0000CA EC?? F???         M                 call    Delay_ms
                     00717                 delay_ms 250            ;wait 250 mSec
0000CE 0EFA              M                 movlw   250
0000D0 EC?? F???         M                 call    Delay_ms
                     00718                 delay_ms 250            ;wait 250 mSec
0000D4 0EFA              M                 movlw   250
0000D6 EC?? F???         M                 call    Delay_ms
0000DA 6EED          00719                 movwf   POSTDEC0,W      ;pop depleted flash count value from stack
0000DC 6EED          00720                 movwf   POSTDEC0,W      ;pop saved flash count value from stack
0000DE 0012          00721                 return
                     00722
                     00723 ;       Flash a countdown starting with WREG times in the LED (1 minimum, 0 means 256 flashes)
                     00724 ;       Uses 7 stack locations (one here and six in FlashLED).
                     00725 ;       Returns with W maintained.
0000E0               00726 FlashCntDn:     ;flash LED Countdown
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

0000E0 6EEC           00727                 movwf   PREINC0         ;push saved countdown value on stack
0000E2 EC?? F???      00728 FlashC1         call    FlashLED        ;flash the LED WReg times
0000E6 4EE8           00729                 dcfsnz  WREG            ;reduce flash counter
0000E8 D???           00730                 bra     FlashCx         ;exit if counter is zero
0000EA D???           00731                 bra     FlashC1         ;loop
0000EC 6EED           00732 FlashCx         movwf   POSTDEC0,W      ;pop saved countdown value from stack
0000EE 0012           00733                 return
                      00734
                      00735
                      00736 ;       Delay the number of uSec specified by the W reg, 1 minimum, 0 means 256 uSec
                      00737 ;       Delay time will be exact if not interrupted.
                      00738 ;       Returns with W reg cleared.
0000F0                00739 Delay_us:                               ;getting here was 2 cycles
0000F0 0000           00740                 nop                     ;1 cycle
0000F2 4EE8           00741 Delayu1         dcfsnz  WREG            ;reduce W ctr ;2 cycles in loop, 1 cycle the last time
0000F4 0012           00742                 return                  ;2 cycles ;return if count was set to 1
0000F6 D???           00743                 bra     $+2             ;2 cycles
0000F8 D???           00744                 bra     Delayu1         ;2 cycles
                      00745
                      00746 ;       Delay the number of 100 uSec intervals specified by the W reg, 1 minimum, 0 means 256*100 uSec
                      00747 ;       Delay time will be exact if not interrupted.
                      00748 ;       Uses 2 stack locations.
                      00749 ;       Returns with W maintained.
0000FA                00750 Delay_hus:                              ;getting here was 2 cycles
0000FA 6EEC           00751                 movwf   PREINC0         ;push saved huSec value on stack, +1 cycle =3
0000FC 6EEC           00752                 movwf   PREINC0         ;push working huSec value on stack, +1 cycle =4
0000FE 4EEF           00753 Delayh1         dcfsnz  INDF0           ;reduce mSec counter; +1 cycle for last time = 5, else 2 cycles
000100 D???           00754                 bra     Delayhx         ;exit if counter is zero; +2 cycles for last time =7
000102 0000           00755                 nop                     ;+1 cycle
000104 0E63           00756                 movlw   99              ;set delay to 99 uSec for intermediate iterations
000106 EC?? F???      00757                 call    Delay_us        ;delay 99 uSecs
00010A D???           00758                 bra     Delayh1         ;retest
00010C 0E62           00759 Delayhx         movlw   98              ;set delay to 98 uSec for last iteration; +1 cycle =8
00010E EC?? F???      00760                 call    Delay_us        ;delay 98 uSecs
000112 6EED           00761                 movwf   POSTDEC0,W      ;pop depleted working mSec value from stack, +1 cycle =9
000114 6EED           00762                 movwf   POSTDEC0,W      ;pop saved mSec value from stack, +1 cycle =10
000116 0012           00763                 return                  ;+2 cycles =12
                      00764
                      00765 ;       Delay the number of mSec specified by the W reg, 1 minimum, 0 means 256 mSec
                      00766 ;       Delay will be slightly greater than specified value. Some loop overhead and
                      00767 ;       interrupt time is not accounted for.
                      00768 ;       Uses 4 stack locations (two here and two in Delay_hus).
                      00769 ;       Returns with W maintained.
000118 6EEC           00770 Delay_ms        movwf   PREINC0         ;push saved mSec value on stack
00011A 6EEC           00771                 movwf   PREINC0         ;push working mSec value on stack
```

Subroutines
LOC OBJECT CODE   LINE SOURCE TEXT
  VALUE

```
00011C 0E0A           00772                movlw   10              ;inner loop delays 1 mSec (10 * 100 uSec)
00011E 4EEF           00773 Delaym1        dcfsnz  INDF0           ;reduce mSec counter
000120 D???           00774                bra     Delaymx         ;exit if counter is zero
000122 EC?? F???      00775                call    Delay_hus       ;delay 1 mSec
000126 D???           00776                bra     Delaym1
000128 6EED           00777 Delaymx        movwf   POSTDEC0,W      ;pop depleted working mSec value from stack
00012A 6EED           00778                movwf   POSTDEC0,W      ;pop saved mSec value from stack
00012C 0012           00779                return
```

LOC OBJECT CODE   LINE SOURCE TEXT
  VALUE

```
                     00780  PAGE
                     00781 ;*******************************************************************************
                     00782                 SUBTITLE "Free Running Timer (Timer1) Driver"
                     00783 ;       This timer keeps track of the amount of time that has occured since the
                     00784 ;       last Software Reset. The timer is started at power on and runs continuously.
                     00785 ;       It is used to create Timestamps for USB messages and Adapter events.
                     00786 ;       The system requirement is to have a Timestamp that can uniquely identify
                     00787 ;       events that take place during extended debug sessions. Since debug sessions
                     00788 ;       can last several hours, the Free Running Timer must be able to span
                     00789 ;       several hours of time without wrapping. To acomplish this, the 16-bit
                     00790 ;       timer counter is extended with 3 bytes of RAM giving it a total size of
                     00791 ;       40 bits. It therefore is capable of spanning about 10 days before wrapping.
                     00792 ;
                     00793 ;       The timer's prescaler is set to divide by 8, so its counter ticks
                     00794 ;       every 0.8 uSec. This results in the 16-bit hardware counter overflowing
                     00795 ;       every 52.4 mSec. When this occurs, an interrupt is generated and the
                     00796 ;       timer's interrupt handler then increments the RAM extension.
                     00797 ;
                     00798 ;       Timestamps (4 byte integers) are formed by reading bits 39 through 8.
                     00799 ;       The lowest order bit if a Timestamp represents 204.8 uSec of time.
                     00800 ;
                     00801 ;       The timer is reset by Software Reset. When this occurs, bits 39 through 0
                     00802 ;       of the timer are set to 0x8000000000 (the most negative number the counter
                     00803 ;       can represent).
                     00804 ;
                     00805 ;       The driver consists of three service routines.
                     00806 ;       1) Initialization subroutine
                     00807 ;       2) Reset subroutine
                     00808 ;       2) Interrupt handler
                     00809 ;       3) Get Timestamp subroutine
                     00810 ;
                     00811 ;*******************************************************************************
                     00812 ;       Free Running Timer (Timer1) initialization subroutine
                     00813 ;       This code runs on the Background thread.
                     00814 ;*******************************************************************************
00012E               00815 Timer1Init:
                     00816 ;       Turn timer off while messing with it
00012E 90CD          00817                 bcf     T1CON,0         ;TMR1ON=0
                     00818 ;       Reset the timer
000130 EC?? F???     00819                 call    Timer1Reset
                     00820 ;       Configure the timer
000134 8ECD          00821                 bsf     T1CON,7         ;RD16=1 sets counter in 16-bit mode
000136 8ACD          00822                 bsf     T1CON,5         ;Set prescaler
000138 88CD          00823                 bsf     T1CON,4         ; to divide by 8
00013A 96CD          00824                 bcf     T1CON,3         ;T1OSCEN=0  turn off to reduce power
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

00013C 94CD           00825                 bcf     T1CON,2         ;T1SYNC#=0  Ignored in this mode
00013E 92CD           00826                 bcf     T1CON,1         ;TMR1CS=0   Use internal clock (FOSC/4)
                      00827 ;       Turn timer on
000140 80CD           00828                 bsf     T1CON,0         ;TMR1ON=1   Start the timer
                      00829 ;       Enable timer interrupts
000142 809D           00830                 bsf     PIE1,TMR1IE
                      00831 ;       return
000144 0012           00832                 return
                      00833 ;
                      00834 ;*****************************************************************************
                      00835 ;       Free Running Timer (Timer1) initialization subroutine
                      00836 ;       This code runs on the Background thread.
                      00837 ;*****************************************************************************
000146                00838 Timer1Reset:
                      00839 ;       Clear the hardware part of the timer
000146 6ACF           00840                 clrf    TMR1H
000148 6ACE           00841                 clrf    TMR1L
                      00842 ;       Set the RAM part of the timer to 0x800000
00014A 6A??           00843                 clrf    FRTIMER_B1
00014C 6A??           00844                 clrf    FRTIMER_B2
00014E 6A??           00845                 clrf    FRTIMER_B3
000150 8E??           00846                 bsf     FRTIMER_B3,7
                      00847 ;       return
000152 0012           00848                 return
                      00849 ;
                      00850 ;*****************************************************************************
                      00851 ;       Free Running Timer (Timer1) Interrupt Handler
                      00852 ;       This code runs on the Low Priority Interrupt Thread.
                      00853 ;       This routine is being executed because the Timer1 interrupt request
                      00854 ;       bit was detected to be on. This bit is turned on when the PIC's Timer1
                      00855 ;       wraps from 0xFFFF to 0x0000.
                      00856 ;       Summary of actions:
                      00857 ;       1) increment the timers RAM extension
                      00858 ;       2) turn off the interrupt request
                      00859 ;*****************************************************************************
000154                00860 Timer1Int:
                      00861 ;       Increment the timer's RAM extension
000154 6AE8           00862                 clrf    WREG
000156 2A??           00863                 incf    FRTIMER_B1
000158 22??           00864                 addwfc  FRTIMER_B2,F
00015A 22??           00865                 addwfc  FRTIMER_B3,F
                      00866 ;       Turn off the interrupt request
00015C 909E           00867                 bcf     PIR1,TMR1IF
                      00868 ;       return
00015E 0012           00869                 return
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00870 ;
                     00871 ;*************************************************************************
                     00872 ;       Get Timestamp subroutine
                     00873 ;       This code may runs on the Low Priority Interrupt Thread or the Background
                     00874 ;       Thread.
                     00875 ;
                     00876 ;(stackInt timestamp) GetTimestamp()                          pushes 10 bytes
                     00877 ;       The caller must push 4 bytes of return parameter space on the software
                     00878 ;       stack (FSR0) for the return value.
                     00879 ;
                     00880 ;       The Free Running Timer is read twice, if both readings are the same,
                     00881 ;       the timestamp is valid. Otherwise, we continue to read the timer
                     00882 ;       until we get two readings that match.
                     00883 ;       If we captured a valid timestamp, then check the interrupt request
                     00884 ;       bit that was recorded with it. If it's on, then add 1 to the RAM extension.
                     00885 ;
                     00886 ;Returns        The four byte value of the Free Runing Timer. The least signigicant
                     00887 ;               byte will be on the top of the stack. The calling routine must pop
                     00888 ;               these four bytes off the stack.
                     00889 ;
                     00890 ;       Register usage: Not permitted due to multithread operation.
                     00891 ;
                     00892 ;       Stack use:
                     00893 ;         -10  (Old top of stack)       <--- Top of stack at entry
                     00894 ;         -9   = Return Timestamp byte 3, highest order
                     00895 ;         -8   = Return Timestamp byte 2
                     00896 ;         -7   = Return Timestamp byte 1
                     00897 ;         -6   = Return Timestamp byte 0 <--- Top of stack on return
                     00898 ;         -5   = Saved WREG
                     00899 ;         -4   = Interrupt pending flag
                     00900 ;       0 to -3  = Temporary Timestamp (2nd copy)
                     00901 ;
000160               00902 GetTimestamp:
                     00903 ;       Get 4 bytes of return result space on stack
000160 6AEC          00904                 clrf    PREINC0
000162 6AEC          00905                 clrf    PREINC0
000164 6AEC          00906                 clrf    PREINC0
000166 6AEC          00907                 clrf    PREINC0
                     00908 ;       Save the WREG on the stack
000168 6EEC          00909                 movwf   PREINC0
                     00910 ;       Get 5 bytes of temporary space on stack
00016A 6AEC          00911                 clrf    PREINC0
00016C 6AEC          00912                 clrf    PREINC0
00016E 6AEC          00913                 clrf    PREINC0
000170 6AEC          00914                 clrf    PREINC0
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

000172 6AEC           00915                 clrf    PREINC0
                      00916 ;Lp:    Read the Free Running Timer into temp storage
000174 0EFF           00917 GetTimestampLp  movlw   -1              ;offset of byte 1
000176 C??? FFEB      00918                 movff   FRTIMER_B1,PLUSW0
00017A 0EFE           00919                 movlw   -2              ;offset of byte 2
00017C C??? FFEB      00920                 movff   FRTIMER_B2,PLUSW0
000180 0EFD           00921                 movlw   -3              ;offset of byte 3 (highest order)
000182 C??? FFEB      00922                 movff   FRTIMER_B3,PLUSW0
000186 50CE           00923                 movf    TMR1L,W         ;These bits are ignored, but need to be accessed
                      00924                                         ; to latch the high order bits.
000188 CFCF FFEF      00925                 movff   TMR1H,INDF0     ;(lowest order of timestamp)
                      00926 ;       Read the interrupt request bit
00018C 0EFC           00927                 movlw   -4
00018E CF9E FFEB      00928                 movff   PIR1,PLUSW0
                      00929 ;       Read the Free Running Timer into return variable
000192 0EF9           00930                 movlw   -7              ;offset of byte 1
000194 C??? FFEB      00931                 movff   FRTIMER_B1,PLUSW0
000198 0EF8           00932                 movlw   -8              ;offset of byte 2
00019A C??? FFEB      00933                 movff   FRTIMER_B2,PLUSW0
00019E 0EF7           00934                 movlw   -9              ;offset of byte 3 (highest order)
0001A0 C??? FFEB      00935                 movff   FRTIMER_B3,PLUSW0
0001A4 50CE           00936                 movf    TMR1L,W         ;These bits are ignored, but need to be accessed
                      00937                                         ; to latch the high order bits.
0001A6 0EFA           00938                 movlw   -6              ;offset of byte 0 (lowest order)
0001A8 CFCF FFEB      00939                 movff   TMR1H,PLUSW0
                      00940 ;       if the two samples are different then goto LP
0001AC 0EFF           00941                 movlw   -1
0001AE CFEB F000      00942                 movff   PLUSW0,FSR0     ;move byte 1 of temp reading to accessable spot
0001B2 0EF9           00943                 movlw   -7              ;offset of byte 1
0001B4 50EB           00944                 movf    PLUSW0,W
0001B6 6200           00945                 cpfseq  FSR0
0001B8 D???           00946                 bra     GetTimestampLp
0001BA 0EFE           00947                 movlw   -2
0001BC CFEB F000      00948                 movff   PLUSW0,FSR0     ;move byte 2 of temp reading to accessable spot
0001C0 0EF8           00949                 movlw   -8              ;offset of byte 2
0001C2 50EB           00950                 movf    PLUSW0,W
0001C4 6200           00951                 cpfseq  FSR0
0001C6 D???           00952                 bra     GetTimestampLp
0001C8 0EFD           00953                 movlw   -3
0001CA CFEB F000      00954                 movff   PLUSW0,FSR0     ;move byte 3 of temp reading to accessable spot
0001CE 0EF7           00955                 movlw   -9              ;offset of byte 3 (highest order)
0001D0 50EB           00956                 movf    PLUSW0,W
0001D2 6200           00957                 cpfseq  FSR0
0001D4 D???           00958                 bra     GetTimestampLp
                      00959 ;       if the sampled interrupt request bit was on then increment the RAM extension
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

0001D6 0EFC           00960                 movlw   -4
0001D8 A0EB           00961                 btfss   PLUSW0,TMR1IF   ;test TMR1IF, on means interrupt requested
0001DA D???           00962                 bra     GetTimestampRtn
0001DC 0EF9           00963                 movlw   -7
0001DE 2AEB           00964                 incf    PLUSW0          ;inc low byte of RAM extension
0001E0 E1??           00965                 bnz     GetTimestampRtn
0001E2 0EF8           00966                 movlw   -8
0001E4 2AEB           00967                 incf    PLUSW0          ;inc middle byte of RAM extension
0001E6 E1??           00968                 bnz     GetTimestampRtn
0001E8 0EF7           00969                 movlw   -9
0001EA 2AEB           00970                 incf    PLUSW0          ;inc high byte of RAM extension
                      00971 ;       Pop the 5 bytes of scratch storage off the stack
0001EC 50ED           00972 GetTimestampRtn movf    POSTDEC0,W
0001EE 50ED           00973                 movf    POSTDEC0,W
0001F0 50ED           00974                 movf    POSTDEC0,W
0001F2 50ED           00975                 movf    POSTDEC0,W
0001F4 50ED           00976                 movf    POSTDEC0,W
                      00977 ;       Restore the WREG from the stack
0001F6 50ED           00978                 movf    POSTDEC0,W
                      00979 ;       return with result on stack
0001F8 0012           00980                 return
```

LOC OBJECT CODE   LINE SOURCE TEXT
  VALUE

```
00981  PAGE
00982 ;*******************************************************************************
00983                 SUBTITLE "Main Background Loop -- USB message receiver"
00984 ;       This routine is the main background loop of the Tape Channel Adapter.
00985 ;       Loop sequence:
00986 ;       1) Check for new USB input data and if the received data completes
00987 ;          a command message, then execute the command.
00988 ;       2) If a DiagUpload Loopback command is active and the Free Running
00989 ;          Timer has exceeded the alarm time, then transmit the event message.
00990 ;       3) If event transmission services has been requested by the Tape
00991 ;          Interface State Machine (TISM), then transmit the event message.
00992 ;
00993 ;       This routine loops looking for Command Message characters coming from
00994 ;       the USB interface. After each message is received, the message command
00995 ;       is examined and the appropriate command handling subroutine is called.
00996 ;       These command handlers are responsible for checking the validity of the
00997 ;       Command Specific Data in the message, executing the command, and sending
00998 ;       the Command Response message.
00999 ;
01000 ;       This thread owns the USB interface. Other threads are not permitted access
01001 ;       to it.
01002 ;
01003 ;       Two USB message buffers are used. One buffer is for receiving Command
01004 ;       Messages and transmitting Command Responses. The other buffer is used to
01005 ;       transmit Event messages. This thread owns the USB Command Buffer.
01006 ;       No interrupt level threads are allowed to access it.
01007 ;
01008 ;       The USB Event Buffer is owned by the Tape Interface State Machine thread
01009 ;       (low priority interrupt thread). When the Tape Interface State Machine
01010 ;       thread wants to send an Event message, it stores the message in the USB
01011 ;       Event Buffer and uses a semaphore to request service from this thread.
01012 ;       The service request gives temporary exclusive read access privileges to
01013 ;       the USB Event Buffer to this thread. When the service is complete, this
01014 ;       thread posts complete in the semaphore and relinquishes its access
01015 ;       privileges.
01016 ;
01017 ;       Once this routine invokes a command handler, service requests from the
01018 ;       Tape Interface State Machine thread will be blocked until the Command
01019 ;       Response message is transmitted.
01020 ;
01021 ;       Similarly, once a Tape Interface State Machine thread service request
01022 ;       is started, execution of Command Messages will be blocked until the
01023 ;       Event Message is transmitted. Note that Command Message characters
01024 ;       will still be accepted and buffered. Only execution of the command is
01025 ;       blocked.
```

LOC OBJECT CODE   LINE SOURCE TEXT
  VALUE

```
                     01026 ;
                     01027 ;       Although called Events, the Command Message Exception Events are processed
                     01028 ;       by this thread through the USB Command Buffer.
                     01029 ;
                     01030 ;       If this thread is blocked from moving a byte into the USB chips transmit
                     01031 ;       FIFO for longer than one minute, then transmission of the message is
                     01032 ;       aborted and the Adapter's status field is updated to indicate this timeout.
                     01033 ;       The occurrence of this timeout is considered a serious error condition.
                     01034 ;       The timeout is also used to prevent thread deadlock.
                     01035 ;
                     01036 ;       The USB Command Buffer is not large enough to hold messages that contain
                     01037 ;       Tape Data. When a message that contains Tape Data is received or transmitted,
                     01038 ;       this data is moved directly to or from the Tape Data buffer. For example,
                     01039 ;       first part of the Download Record messages is stored in the USB Command
                     01040 ;       Buffer and the other part is stored in the Tape Data Buffer.
                     01041 ;
                     01042 ;       The Tape Data Buffer is owned by the Tape Interface State Machine thread.
                     01043 ;       The Tape Interface State Machine thread temporary grants read/write
                     01044 ;       permission to the Tape Data Buffer to this thread when it needs to
                     01045 ;       receive or transmit tape data. If a Command Message attempts to access
                     01046 ;       this buffer without permission the command will be aborted and
                     01047 ;       this will be noted in the resulting Command Response Message. Diagnostic
                     01048 ;       Command Messages bypass the access permission check.
                     01049 ;
                     01050 ;       Register usage:
                     01051 ;        REG0 = byte received from USB chip
                     01052 ;        REG1 = parameters for command/event handler subroutines
                     01053 ;
                     01054 ;*******************************************************************************
0001FA               01055 Main:
                     01056 ;===============================================================================
                     01057 ;***    Check for and handle USB input
                     01058 ;===============================================================================
                     01059 ;       if the usb interface presenting a character
0001FA EC?? F???     01060                 call    GetUsbChar      ;STATUS_C==1 if char; WReg=char
0001FE E2??          01061                 bc      $+2
000200 EF?? F???     01062                 goto    MainEndif00
                     01063 ;         Reg0 = usb character
000204 6E??          01064                 movwf   REG0
                     01065 ;***    Start Flag Section ************************************************
                     01066 ;         if USB state == usbSt_ExpBegin
000206 0E00          01067                 movlw   usbSt_ExpBegin
000208 62??          01068                 cpfseq  usbState
00020A D???          01069                 bra     MainEndifBeg
                     01070 ;           if usb character is the start flag
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

00020C 0EF0           01071                 movlw   USB_START_FLAG
00020E 62??           01072                 cpfseq  REG0
000210 D???           01073                 bra     MainBeginElse
                      01074 ;             Clear the usb status flags to start fresh for this message
000212 6A??           01075                 clrf    usbStatus
                      01076 ;             Clear the USB working CRC
000214 6A??           01077                 clrf    usbCRC
                      01078 ;             Set the USB command to "not received"
000216 6A??           01079                 clrf    usbCommand
                      01080 ;             Initialize usb buffer pointer to the beginning of buffer
000218 EE25 F000      01081                 lfsr    FSR2,uBuffer
00021C CFD9 F???      01082                 movff   FSR2L,usbRecNxt
000220 CFDA F???      01083                 movff   FSR2H,usbRecNxt+1
                      01084 ;             Initialize usb tape data buffer pointer to the beginning of buffer
000224 EE26 F000      01085                 lfsr    FSR2,tBuffer
000228 CFD9 F???      01086                 movff   FSR2L,usbTapeDataNxt
00022C CFDA F???      01087                 movff   FSR2H,usbTapeDataNxt+1
                      01088 ;             Initialize the count of command specific data
000230 6A??           01089                 clrf    usbCountH
000232 6A??           01090                 clrf    usbCountL
                      01091 ;             Set USB state = usbSt_ExpLen1
000234 0E01           01092                 movlw   usbSt_ExpLen1
000236 6E??           01093                 movwf   usbState
                      01094 ;           endif usb char is the start flag
000238 D???           01095                 bra     MainEndif01     ;exit state decode ifelse tree
                      01096 ;           else usb char is not the start flag
00023A                01097 MainBeginElse:
                      01098 ;               remember that non flag characters were received
00023A 88??           01099                 bsf     ANLS_STATUS,ANLS_UsbFlag        ;mark in analyzer status
                      01100 ;             ignore character and keep waiting for start flag
00023C D???           01101                 bra     MainEndif01     ;exit state decode ifelse tree
                      01102 ;           endelse usb char is not the start flag
                      01103 ;         endif state==usbSt_ExpBegin
00023E D???           01104                 bra     MainEndif01
000240                01105 MainEndifBeg:
                      01106 ;***    Length Low Section ************************************************
                      01107 ;         elseif USB state == usbSt_ExpLen1
000240 0E01           01108                 movlw   usbSt_ExpLen1
000242 62??           01109                 cpfseq  usbState
000244 D???           01110                 bra     MainEndifLen1
                      01111 ;           Add the byte to the CRC
000246 50??           01112                 movf    REG0,W
000248 26??           01113                 addwf   usbCRC
                      01114 ;           Save this byte of length
00024A 6E??           01115                 movwf   usbLengthL
```

LOC OBJECT CODE   LINE SOURCE TEXT
  VALUE

```
                     01116 ;           Set USB state = usbSt_ExpLen2
00024C 0E02          01117                 movlw   usbSt_ExpLen2
00024E 6E??          01118                 movwf   usbState
                     01119 ;         endif state==usbSt_ExpBegin
000250 D???          01120                 bra     MainEndif01
000252               01121 MainEndifLen1:
                     01122 ;***    Length High Section ************************************************
                     01123 ;         elseif USB state == usbSt_ExpLen2
000252 0E02          01124                 movlw   usbSt_ExpLen2
000254 62??          01125                 cpfseq  usbState
000256 D???          01126                 bra     MainEndifLen2
                     01127 ;           Add the byte to the CRC
000258 50??          01128                 movf    REG0,W
00025A 26??          01129                 addwf   usbCRC
                     01130 ;           Save this byte of length
00025C 6E??          01131                 movwf   usbLengthH
                     01132 ;           Set USB state = usbSt_ExpCmd
00025E 0E03          01133                 movlw   usbSt_ExpCmd
000260 6E??          01134                 movwf   usbState
                     01135 ;           if msg_length == 0 then Set USB state = usbSt_ExpCRC
000262 52??          01136                 movf    usbLengthL
000264 E1??          01137                 bnz     MainEndifLen2x
000266 52??          01138                 movf    usbLengthH
000268 E1??          01139                 bnz     MainEndifLen2x
00026A 0E05          01140                 movlw   usbSt_ExpCRC
00026C 6E??          01141                 movwf   usbState
                     01142 ;         endif state==usbSt_ExpLen2
00026E D???          01143 MainEndifLen2x  bra     MainEndif01
000270               01144 MainEndifLen2:
                     01145 ;***    Command Section ****************************************************
                     01146 ;         elseif USB state == usbSt_ExpCmd
000270 0E03          01147                 movlw   usbSt_ExpCmd
000272 62??          01148                 cpfseq  usbState
000274 D???          01149                 bra     MainEndifCmd
                     01150 ;           increment the number of command specific data byte received
000276 6AE8          01151                 clrf    WREG
000278 2A??          01152                 incf    usbCountL
00027A 22??          01153                 addwfc  usbCountH
                     01154 ;           Add the byte to the CRC
00027C 50??          01155                 movf    REG0,W
00027E 26??          01156                 addwf   usbCRC
                     01157 ;           Save the command
000280 6F00          01158                 movwf   usbCommand,F
                     01159 ;           Store the byte in the USB data buffer
000282 C??? FFD9     01160                 movff   usbRecNxt,FSR2L
```

LOC OBJECT CODE   LINE SOURCE TEXT
  VALUE

```
000286 C??? FFDA      01161                movff   usbRecNxt+1,FSR2H
00028A 6EDE           01162                movwf   POSTINC2
00028C CFD9 F???      01163                movff   FSR2L,usbRecNxt
000290 CFDA F???      01164                movff   FSR2H,usbRecNxt+1
                      01165 ;           Set USB state = usbSt_ExpData
000294 0E04           01166                movlw   usbSt_ExpData
000296 6E??           01167                movwf   usbState
                      01168 ;           if msg_length == command specific data byte count then Set USB state = usbSt_ExpCRC
000298 52??           01169                movf    usbLengthL
00029A 5E??           01170                subwf   usbCountL
00029C E1??           01171                bnz     MainEndifCmdx
00029E 52??           01172                movf    usbLengthH
0002A0 5E??           01173                subwf   usbCountH
0002A2 E1??           01174                bnz     MainEndifCmdx
0002A4 0E05           01175                movlw   usbSt_ExpCRC
0002A6 6E??           01176                movwf   usbState
                      01177 ;         endif state==uusbSt_ExpData
0002A8 D???           01178 MainEndifCmdx   bra     MainEndif01
0002AA                01179 MainEndifCmd:
                      01180 ;***    Command Data Section ************************************************
                      01181 ;         elseif USB state == usbSt_ExpData
0002AA 0E04           01182                movlw   usbSt_ExpData
0002AC 62??           01183                cpfseq  usbState
0002AE D???           01184                bra     MainEndifData
                      01185 ;           increment the number of command specific data byte received
0002B0 6AE8           01186                clrf    WREG
0002B2 2A??           01187                incf    usbCountL
0002B4 22??           01188                addwfc  usbCountH
                      01189 ;           add the byte to the CRC
0002B6 50??           01190                movf    REG0,W
0002B8 26??           01191                addwf   usbCRC
                      01192 ;           if (destination != tape buffer
                      01193 ;             && (command==usbCmd_DnldRecord && usbCount==3)
                      01194 ;                || (command==usbCmd_DiagUpldLp && usbCount==7))
0002BA BE??           01195                btfsc   usbStatus,usbSs_StTape
0002BC D???           01196                bra     MainDataDiagFalse
0002BE 0E09           01197                movlw   usbCmd_DnldRecord
0002C0 62??           01198                cpfseq  usbState
0002C2 D???           01199                bra     MainDataDnldFalse
0002C4 0E02           01200                movlw   2
0002C6 62??           01201                cpfseq  usbCountL
0002C8 D???           01202                bra     MainDataDnldFalse
0002CA D???           01203                bra     MainDataDnldTrue
0002CC 0E10           01204 MainDataDnldFalse movlw usbCmd_DiagUpldLp
0002CE 62??           01205                cpfseq  usbState
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

0002D0 D???           01206                 bra     MainDataDiagFalse
0002D2 0E06           01207                 movlw   6
0002D4 62??           01208                 cpfseq  usbCountL
0002D6 D???           01209                 bra     MainDataDiagFalse
                      01210 ;             Switch destination buffer to the Tape Data buffer
0002D8 8E??           01211 MainDataDnldTrue bsf    usbStatus,usbSs_StTape
                      01212 ;           endif
0002DA                01213 MainDataDiagFalse:
                      01214 ;           if destination is the Tape Data buffer
0002DA AE??           01215                 btfss   usbStatus,usbSs_StTape
0002DC D???           01216                 bra     MainDataStoreUsb
                      01217 ;             if buffer has room (next==end+1)
0002DE 0E01           01218                 movlw   (tBufferEnd+1)&&0xff    ;compare low order byte
0002E0 62??           01219                 cpfseq  usbTapeDataNxt
0002E2 D???           01220                 bra     MainDataTapeOk
0002E4 0E01           01221                 movlw   (tBufferEnd+1)>>8&&0xff ;compare high order byte
0002E6 62??           01222                 cpfseq  usbTapeDataNxt+1
0002E8 D???           01223                 bra     MainDataTapeOk
0002EA D???           01224                 bra     MainDataTapeFull
                      01225 ;               Store the byte in the Tape Data buffer
0002EC C??? FFD9      01226 MainDataTapeOk    movff usbTapeDataNxt,FSR2L
0002F0 C??? FFDA      01227                   movff usbTapeDataNxt+1,FSR2H
0002F4 C??? FFDE      01228                   movff REG0,POSTINC2
0002F8 CFD9 F???      01229                   movff FSR2L,usbTapeDataNxt
0002FC CFDA F???      01230                   movff FSR2H,usbTapeDataNxt+1
                      01231 ;             endif
000300 D???           01232                 bra     MainDataTapeEndelse
                      01233 ;             else Tape buffer does not have room
                      01234 ;               Set buffer overflow usb status
000302 88??           01235 MainDataTapeFull  bsf   usbStatus,usbSs_BufOflw
                      01236 ;             endelse
000304 D???           01237                 bra     MainDataEndelse
000306                01238 MainDataTapeEndelse:
                      01239 ;           endif
000306 D???           01240                 bra     MainDataEndelse
                      01241 ;           else destination is not the Tape Data buffer
000308                01242 MainDataStoreUsb:
                      01243 ;             if buffer has room (next==end+1)
000308 EE25 F080      01244                 lfsr    FSR2,uBufferEnd+1
00030C 50D9           01245                 movf    FSR2L,W                 ;compare low order byte
00030E 62??           01246                 cpfseq  usbRecNxt
000310 D???           01247                 bra     MainDataUsbOk
000312 50DA           01248                 movf    FSR2H,W                 ;compare high order byte
000314 62??           01249                 cpfseq  usbRecNxt+1
000316 D???           01250                 bra     MainDataUsbOk
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

000318 D???           01251                 bra     MainDataUsbFull
                      01252 ;               Store the byte in the USB data buffer
00031A C??? FFD9      01253 MainDataUsbOk     movff usbRecNxt,FSR2L
00031E C??? FFDA      01254                   movff usbRecNxt+1,FSR2H
000322 C??? FFDE      01255                   movff REG0,POSTINC2
000326 CFD9 F???      01256                   movff FSR2L,usbRecNxt
00032A CFDA F???      01257                   movff FSR2H,usbRecNxt+1
                      01258 ;             endif
00032E D???           01259                 bra     MainDataUsbEndelse
                      01260 ;             else Tape buffer does not have room
                      01261 ;               Set buffer overflow usb status
000330 88??           01262 MainDataUsbFull  bsf    usbStatus,usbSs_BufOflw
                      01263 ;             endelse
000332                01264 MainDataUsbEndelse:
                      01265 ;           endelse
000332                01266 MainDataEndelse:
                      01267 ;           if msg_length == command specific data byte count then Set USB state = usbSt_ExpCRC
000332 52??           01268                 movf    usbLengthL
000334 5E??           01269                 subwf   usbCountL
000336 E1??           01270                 bnz     MainEndifDatax
000338 52??           01271                 movf    usbLengthH
00033A 5E??           01272                 subwf   usbCountH
00033C E1??           01273                 bnz     MainEndifDatax
00033E 0E05           01274                 movlw   usbSt_ExpCRC
000340 6E??           01275                 movwf   usbState
                      01276 ;         endif state==uusbSt_ExpData
000342 D???           01277 MainEndifDatax  bra     MainEndif01
000344                01278 MainEndifData:
                      01279 ;***    CRC Section ************************************************************
                      01280 ;         elseif USB state == usbSt_ExpCRC
000344 0E05           01281                 movlw   usbSt_ExpCRC
000346 62??           01282                 cpfseq  usbState
000348 D???           01283                 bra     MainEndifCRC
                      01284 ;           Add the byte to the CRC
00034A 50??           01285                 movf    REG0,W
00034C 26??           01286                 addwf   usbCRC
                      01287 ;           if the crc is not zero, then set the Bad CRC flag
00034E E0??           01288                 bz      MainGoodCRC
000350 84??           01289                 bsf     usbStatus,usbSs_BadCRC
                      01290 ;           Save the CRC character in REG1
000352 6E??           01291 MainGoodCRC             movwf   REG1
                      01292 ;         endif state==usbSt_ExpCRC
000354 D???           01293                 bra     MainEndif01
000356                01294 MainEndifCRC:
                      01295 ;***    Stop Flag Section ******************************************************
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     01296 ;         elseif USB state == usbSt_ExpStop
000356 0E06          01297                movlw   usbSt_ExpStop
000358 62??          01298                cpfseq  usbState
00035A D???          01299                bra     MainEndifStop
                     01300 ;           if usb character is the stop flag
00035C 0EF1          01301                movlw   USB_STOP_FLAG
00035E 62??          01302                cpfseq  REG0
000360 D???          01303                bra     MainNotStop
                     01304 ;             if the received crc was bad ....................................
000362 A4??          01305                  btfss   usbStatus,usbSs_BadCRC
000364 D???          01306                  bra     MainStopCrcOk
                     01307 ;              REG0 = expected CRC (usbCRC.xor.REG1)
000366 50??          01308                  movf    usbCRC,W
000368 18??          01309                  xorwf   REG1,W
00036A 6E??          01310                  movwf   REG0
                     01311 ;              respond with a CRC error event
00036C EC?? F???     01312                  call    RespondCrcErr   ;REG0=expected CRC, REG1=rcvd CRC
000370 D???          01313                  bra     MainStopLv1     ;exit ifelse tree
                     01314 ;             endif
                     01315 ;             elseif command==usbCmd_DefVirtDrv ..............................
000372 0E04          01316 MainStopCrcOk   movlw   usbCmd_DefVirtDrv
000374 62??          01317                  cpfseq  usbCommand
000376 D???          01318                  bra     MainNotDefVirtDrv
                     01319 ;              perform "define virttual drive" command
000378 EC?? F???     01320                  call    CmdDefVirtDrv
00037C D???          01321                  bra     MainStopLv1     ;exit ifelse tree
                     01322 ;             endelseif
                     01323 ;             elseif command==usbCmd_DiagUpldLp ..............................
00037E 0E10          01324 MainNotDefVirtDrv movlw usbCmd_DiagUpldLp
000380 62??          01325                  cpfseq  usbCommand
000382 D???          01326                  bra     MainNotDiagUpldLp
                     01327 ;              perform "diag upload loop" command
000384 EC?? F???     01328                  call    CmdDiagUpldLp
000388 D???          01329                  bra     MainStopLv1     ;exit ifelse tree
                     01330 ;             endelseif
                     01331 ;             elseif command==usbCmd_UpldRecord ..............................
00038A 0E08          01332 MainNotDiagUpldLp movlw usbCmd_UpldRecord
00038C 62??          01333                  cpfseq  usbCommand
00038E D???          01334                  bra     MainNotUpldRecord
                     01335 ;              perform "upload record" command
                     01336                  ;call   CmdUpldRecord
000390 D???          01337                  bra     MainStopLv1     ;exit ifelse tree
                     01338 ;             endelseif
000392               01339 MainNotUpldRecord:
                     01340 ;             else invalid command ...........................................
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     01341 ;               respond with a invalid command code event
                     01342                 ;call    RespondInvCmd
                     01343 ;             endelse
000392               01344 MainStopLv1:
                     01345 ;           endif stop flag
                     01346 ;           else not stop flag when it should have been
000392               01347 MainNotStop:
                     01348 ;             respond with a invalid command code event
000392 EC?? F???     01349                 call    RespondTrunc    ;send truncated message event
                     01350 ;             set receive state to purge incomming characters until stop flag is received
000396 0E07          01351                 movlw   usbSt_ExpPurge
000398 52??          01352                 movf    usbState
                     01353 ;           endelse not stop flag
                     01354 ;         endif state==usbSt_ExpStop
00039A D???          01355                 bra     MainEndif01
00039C               01356 MainEndifStop:
                     01357 ;***    Purge Section ************************************************************
                     01358 ;         elseif USB state == usbSt_ExpStop
00039C 0E07          01359                 movlw   usbSt_ExpPurge
00039E 62??          01360                 cpfseq  usbState
0003A0 D???          01361                 bra     MainEndifPurge
                     01362 ;           if usb character is the stop flag
0003A2 0EF1          01363                 movlw   USB_STOP_FLAG
0003A4 62??          01364                 cpfseq  REG0
0003A6 D???          01365                 bra     MainEndifPurge
                     01366 ;             set receive state to expect a start flag
0003A8 0E00          01367                 movlw   usbSt_ExpBegin
0003AA 52??          01368                 movf    usbState
                     01369 ;           endif
0003AC D???          01370                 bra     MainEndif01
                     01371 ;           else keep purging
                     01372 ;         endelseif
0003AE               01373 MainEndifPurge:
                     01374 ;***    Invalid State Section ********************************************
                     01375 ;         else not in a legal state
                     01376                 ;hang 1
                     01377 ;         end else not in a legal state
0003AE               01378 MainEndif01:
                     01379 ;       endif the usb is presenting a character
0003AE               01380 MainEndif00:
                     01381 ;=============================================================================
                     01382 ;***    Check for Diagnostic Alarm ***************************************
                     01383 ;=============================================================================
                     01384 ;       if DiagUpload Loopback is active
0003AE A0??          01385                 btfss   diagCntl,diag_UpLdLb
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

0003B0 D???           01386                bra     MainIf2x
                      01387 ;         if current timestamp >= alarm time
0003B2 EC?? F???      01388                call    GetTimestamp            ;Time is on stack
0003B6 50??           01389                movf    diagAlarmTime,W
0003B8 5CED           01390                subwf   POSTDEC0,W              ;Current time - Alarm time
0003BA 50??           01391                movf    diagAlarmTime+1,W
0003BC 58ED           01392                subwfb  POSTDEC0,W
0003BE 50??           01393                movf    diagAlarmTime+2,W
0003C0 58ED           01394                subwfb  POSTDEC0,W
0003C2 50??           01395                movf    diagAlarmTime+3,W
0003C4 58ED           01396                subwfb  POSTDEC0,W
0003C6 E6??           01397                bn      MainIf4x
                      01398 ;           Fake a TISM service request
0003C8 6A??           01399                clrf    EDS_SREQ        ;the order of these instructions is important
0003CA 6A??           01400                clrf    EDS_RCODE
0003CC EE25 F080      01401                lfsr    FSR2,eBuffer
0003D0 CFDF F???      01402                movff   INDF2,EDS_SREQ
                      01403 ;           if diagnostic repeat count == 0
0003D4 66??           01404                tstfsz  diagRepeatCnt
0003D6 D???           01405                bra     MainIf5f
0003D8 66??           01406                tstfsz  diagRepeatCnt+1
0003DA D???           01407                bra     MainIf5f
                      01408 ;             Terminate the diagnostic instruction
0003DC 90??           01409                bcf     diagCntl,diag_UpLdLb
                      01410 ;           endif diagnostic repeat count == 0
0003DE D???           01411                bra     MainIf5x
                      01412 ;           else diagnostic repeat count != 0
                      01413 ;             New alarm time = current time + repeat delay time
0003E0 EC?? F???      01414 MainIf5f       call    GetTimestamp            ;Time is on stack
0003E4 50ED           01415                movf    POSTDEC0,W
0003E6 24??           01416                addwf   diagDelayTime,W
0003E8 6E??           01417                movwf   diagAlarmTime
0003EA 50ED           01418                movf    POSTDEC0,W
0003EC 20??           01419                addwfc  diagDelayTime+1,W
0003EE 6E??           01420                movwf   diagAlarmTime+1
0003F0 50ED           01421                movf    POSTDEC0,W
0003F2 20??           01422                addwfc  diagDelayTime+2,W
0003F4 6E??           01423                movwf   diagAlarmTime+2
0003F6 50ED           01424                movf    POSTDEC0,W
0003F8 20??           01425                addwfc  diagDelayTime+3,W
0003FA 6E??           01426                movwf   diagAlarmTime+3
                      01427 ;             Decriment the repeat count
0003FC 06??           01428                decf    diagRepeatCnt
0003FE E3??           01429                bnc     MainIf5x
000400 06??           01430                decf    diagRepeatCnt+1
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     01431 ;           endelse diagnostic repeat count != 0
000402               01432 MainIf5x
                     01433 ;         endif current timestamp >= alarm time
000402               01434 MainIf4x:
                     01435 ;       endif DiagUpload Loopback is active
000402               01436 MainIf2x:
                     01437 ;================================================================================
                     01438 ;***    Check for TISR Service Request ****************************************
                     01439 ;================================================================================
                     01440 ;       if EDS_RCODE==0 && EDS_SREQ!=0  ;checking order is important
000402 50??          01441                 movf    EDS_RCODE,W
000404 E0??          01442                 bz      MainIf3x
000406 50??          01443                 movf    EDS_SREQ,W
000408 E1??          01444                 bnz     MainIf3x
                     01445 ;         Transmit the event message
00040A EC?? F???     01446                 call    XmitEvent
                     01447 ;         Post complete in the semaphore
00040E 0E01          01448                 movlw   1
000410 52??          01449                 movf    EDS_RCODE,F
                     01450 ;       endif EDS_RCODE==0 && EDS_SREQ!=0
000412               01451 MainIf3x:
                     01452 ;       display the usb state in the low order 3 bits if the direction register
                     01453
                     01454 ;       goto the top of the Main loop
000412 EF?? F???     01455                 goto    Main
                     01456
                     01457 ;*****************************************************************************
                     01458                 SUBTITLE "Main loop subroutines"
                     01459 ;
                     01460 ;(WReg=char,CC.C=bool) GetUsbChar()
                     01461 ;Returns        WReg, if there is an input character from the usb the usb port,
                     01462 ;                       it is returned in the WReg. If not, the WReg is invalid.
                     01463 ;               CC,C  The carry bit if the condition code is set if a character
                     01464 ;                       was received. Else the carry bit is cleared.
                     01465 ;
                     01466 ;       If the is_simulating assembler variable is set, then then code is
                     01467 ;       inserted to return the next dummy command character, else the USB port
                     01468 ;       is really checked for data.
                     01469 ;
                     01470 ;       Destroys: TBLPTR
                     01471 ;
000416               01472 GetUsbChar:
                     01473         if is_simulating
                     01474 ;       This code is for simulation debug only
                     01475 ;       if      usbSimNxt>UsbFakeE
```

LOC OBJECT CODE   LINE SOURCE TEXT
  VALUE

```
                     01476                 movlw   (UsbFakeE>>16)&0xff     ;compare high order
                     01477                 subwf   usbSimNxt+2,W           ;
                     01478                 bn      GetUsbCharPass
                     01479                 bnz     GetUsbCharFail
                     01480                 movlw   (UsbFakeE>>8)&0xff      ;compare middle order
                     01481                 subwf   usbSimNxt+1,W           ;
                     01482                 bn      GetUsbCharPass
                     01483                 bnz     GetUsbCharFail
                     01484                 movlw   UsbFakeE&0xff           ;compare low order
                     01485                 subwf   usbSimNxt,W             ;
                     01486                 bn      GetUsbCharPass
                     01487                 bz      GetUsbCharPass
                     01488 ;       then return with "no char" indication
                     01489 GetUsbCharFail: bcf     STATUS,C                ;clear the carry bit
                     01490                 return
                     01491 ;       endif
                     01492 GetUsbCharPass:
                     01493 ;       receive the next character
                     01494                 movff   usbSimNxt+2,TBLPTRU
                     01495                 movff   usbSimNxt+1,TBLPTRH
                     01496                 movff   usbSimNxt,TBLPTRL
                     01497                 tblrd   *+                      ;read the table byte then increment
                     01498                 movff   TBLPTRU,usbSimNxt+2     ;store table pointer back
                     01499                 movff   TBLPTRH,usbSimNxt+1
                     01500                 movff   TBLPTRL,usbSimNxt
                     01501                 ; WReg still contains character
                     01502                 bsf     STATUS,C                ;set the carry bit
                     01503 ;       return
                     01504                 return
                     01505         endif   ; if is_simulating
```

LOC OBJECT CODE   LINE SOURCE TEXT
  VALUE

```
                     01506  PAGE
                     01507 ;*************************************************************************
                     01508 ;       Handle the "Define Virtual Drive"  command message      pushes 5 bytes
                     01509 ;  void CmdDefVirtDrv()
                     01510 ;       Define/Redefine a virtual tape drive by setting its state per the
                     01511 ;       command specific data supplied with this command and enabling emulation.
                     01512 ;
                     01513 ;       Command specific data: (in usb buffer)
                     01514 ;        0 = tape drive address
                     01515 ;        1 = tape drive type
                     01516 ;        2 = drive state flags
                     01517 ;        3 = LRC
                     01518 ;        4 = tape position (low order)
                     01519 ;        5 = tape position (low order)
                     01520 ;
                     01521 ;       Response data:
                     01522 ;        0 = return code, one of the below codes will be returned
                     01523 ;               RC_SUCCESS, RC_BAD_ADDR
                     01524 ;
                     01525 ;       REG usage:
                     01526 ;       REG0-5 initially used as scratch space, then ...
                     01527 ;       REG0 = Command/Event code (usbCmd_DefVirtDrv)
                     01528 ;       REG1 = number of bytes to be sent from USB xmit buffer (1)
                     01529 ;       REG2,3 = number of bytes to be sent from tape buffer (zero for this command)
                     01530 ;
                     01531 ;*************************************************************************
000416               01532 CmdDefVirtDrv:
                     01533 ;       Save the WREG on the stack
000416 6EEC          01534                 movwf   PREINC0
                     01535 ;       Save REG0 through REG3 on stack
000418 C??? FFEC     01536                 movff   REG0,PREINC0
00041C C??? FFEC     01537                 movff   REG1,PREINC0
000420 C??? FFEC     01538                 movff   REG2,PREINC0
000424 C??? FFEC     01539                 movff   REG3,PREINC0
000428 C??? FFEC     01540                 movff   REG4,PREINC0
00042C C??? FFEC     01541                 movff   REG5,PREINC0
                     01542 ;       Save FSR2 on stack
000430 CFD9 FFEC     01543                 movff   FSR2L,PREINC0
000434 CFDA FFEC     01544                 movff   FSR2H,PREINC0
                     01545 ;       if drive address is valid
000438 EE25 F000     01546                 lfsr    FSR2,uBuffer
00043C 50DE          01547                 movf    POSTINC2,W
00043E 6E??          01548                 movwf   REG0
000440 EC?? F???     01549                 call    IsAddrValid
000444 E1??          01550                 bnz     CmdDefVirtDrvIf1f
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     01551 ;         Move new drive state into registers
000446 CFDE F???     01552                 movff   POSTINC2,REG1           ;DS_TYPE
00044A CFDE F???     01553                 movff   POSTINC2,REG2           ;DSTATE
00044E CFDE F???     01554                 movff   POSTINC2,REG3           ;DS_LRC
000452 CFDE F???     01555                 movff   POSTINC2,REG4           ;DS_TAPEPOS low
000456 CFDE F???     01556                 movff   POSTINC2,REG5           ;DS_TAPEPOS high
                     01557 ;         Get addressability to the specified drive's area
00045A EE?? F0??     01558                 lfsr    FSR2,drive1State-8      ;-8 for 0 origin
00045E 50??          01559                 movf    REG0,W                  ;drive address
000460 0B07          01560                 andlw   0x07
000462 0D08          01561                 mullw   8                       ;drive address * 8
000464 26D9          01562                 addwf   FSR2L                   ;add to FSR2 base addr
000466 6AE8          01563                 clrf    WREG
000468 22DA          01564                 addwfc  FSR2H
                     01565 ;         Move the received state data to the appropriate drive area
00046A C??? FFDE     01566                 movff   REG2,POSTINC2           ;DSTATE
00046E C??? FFDE     01567                 movff   REG3,POSTINC2           ;DS_LRC
000472 C??? FFDE     01568                 movff   REG4,POSTINC2           ;DS_TAPEPOS low
000476 C??? FFDE     01569                 movff   REG5,POSTINC2           ;DS_TAPEPOS high
00047A C??? FFDE     01570                 movff   REG1,POSTINC2           ;DS_TYPE
                     01571 ;         Convert the drive address to a bit mask
00047E 50??          01572                 movf    REG0,W                  ;drive address
000480 EC?? F???     01573                 call    Addr2Mask               ;convert
                     01574 ;         Enable the device
000484 12??          01575                 iorwf   ANLS_MMASK,F
000486 12??          01576                 iorwf   ANLS_EMASK,F
                     01577 ;         Set response return code = RC_SUCCESS
000488 EE25 F000     01578                 lfsr    FSR2,uBuffer
00048C 0E00          01579                 movlw   RC_SUCCESS
00048E 6EDF          01580                 movwf   INDF2
                     01581 ;       endif
000490 D???          01582                 bra     CmdDefVirtDrvIf1x
                     01583 ;       else drive address is not valid
                     01584 ;         Set response return code = RC_BAD_ADDR
000492 EE25 F000     01585 CmdDefVirtDrvIf1f lfsr  FSR2,uBuffer
000496 0E01          01586                 movlw   RC_BAD_ADDR
000498 6EDF          01587                 movwf   INDF2
                     01588 ;       endelse
00049A               01589 CmdDefVirtDrvIf1x:
                     01590 ;       Set up event code
00049A 0E04          01591                 movlw   usbCmd_DefVirtDrv
00049C 52??          01592                 movf    REG0,F
                     01593 ;       Initialize the number of bytes to be transfered from the USB buffer = 1
00049E 0E01          01594                 movlw   1
0004A0 52??          01595                 movf    REG1,F
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                      01596 ;       Initialize the number of bytes to be transfered from the Tape Date buffer = 0
0004A2 6A??           01597                 clrf    REG2
0004A4 6A??           01598                 clrf    REG3
                      01599 ;       Send the message
0004A6 EC?? F???      01600                 call    SendUsbMsg
                      01601 ;Rtn:   Restore saved registers
0004AA CFED FFDA      01602 CmdDefVirtDrvRtn movff  POSTDEC0,FSR2H
0004AE CFED FFD9      01603                 movff   POSTDEC0,FSR2L
0004B2 CFED F???      01604                 movff   POSTDEC0,REG5
0004B6 CFED F???      01605                 movff   POSTDEC0,REG4
0004BA CFED F???      01606                 movff   POSTDEC0,REG3
0004BE CFED F???      01607                 movff   POSTDEC0,REG2
0004C2 CFED F???      01608                 movff   POSTDEC0,REG1
0004C6 CFED F???      01609                 movff   POSTDEC0,REG0
                      01610 ;       Restore the WREG
0004CA 50ED           01611                 movf    POSTDEC0,W
                      01612 ;       return
0004CC 0012           01613                 return
                      01614
```

LOC OBJECT CODE   LINE SOURCE TEXT
  VALUE

```
                      01615  PAGE
                      01616 ;*********************************************************************
                      01617 ;       Handle the "DiagUpload Loopback" command message        pushes 9 bytes
                      01618 ;  void CmdDiagUpldLp()
                      01619 ;       The prepared tape record has been into the Tape Data buffer.
                      01620 ;       Instruct the adapter to:
                      01621 ;       1) Disable monitoring and emulation of all drives except the addressed drive.
                      01622 ;       2) Wait for initial_delay seconds.
                      01623 ;       3) Pretend that the 1401 just completed writing the stored data to
                      01624 ;          addressed drive.
                      01625 ;       4) If repeat_count is non-zero, then repeat pretending to receive this
                      01626 ;          record repeat_count times delaying repeat_ delay between repetitions.
                      01627 ;       Define/Redefine a virtual tape drive by setting its state per the
                      01628 ;       command specific data supplied with this command and enabling emulation.
                      01629 ;
                      01630 ;       Command specific data: (in usb buffer)
                      01631 ;        0 = tape drive address
                      01632 ;        1 = initial delay 0 (lowest order)
                      01633 ;        2 = initial delay 1
                      01634 ;        3 = initial delay 2
                      01635 ;        4 = initial delay 3
                      01636 ;        5 = repeat delay 0 (lowest order)
                      01637 ;        6 = repeat delay 1
                      01638 ;        7 = repeat delay 2
                      01639 ;        8 = repeat delay 3
                      01640 ;        9 = repeat count low
                      01641 ;        10= repeat count high
                      01642 ;
                      01643 ;       Command specific data: (in tape data buffer)
                      01644 ;        0 - end = tape data to be sent back to the pc
                      01645 ;
                      01646 ;       Response data
                      01647 ;        0 = return code, one of the below codes will be returned
                      01648 ;               RC_SUCCESS, RC_BAD_ADDR
                      01649 ;
                      01650 ;       REG usage:
                      01651 ;       Initially REG0-REG5 are used for scratch purposes.
                      01652 ;       REG0 = Command/Event code (usbCmd_DefVirtDrv)
                      01653 ;       REG1 = number of bytes to be sent from USB xmit buffer (1)
                      01654 ;       REG2,3 = number of bytes to be sent from tape buffer (zero for this command)
                      01655 ;
                      01656 ;*********************************************************************
0004CE                01657 CmdDiagUpldLp:
                      01658 ;       Save the WREG on the stack
0004CE 6EEC           01659                 movwf   PREINC0
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     01660 ;       Save REG0 through REG3 on stack
0004D0 C??? FFEC     01661                 movff   REG0,PREINC0
0004D4 C??? FFEC     01662                 movff   REG1,PREINC0
0004D8 C??? FFEC     01663                 movff   REG2,PREINC0
0004DC C??? FFEC     01664                 movff   REG3,PREINC0
0004E0 C??? FFEC     01665                 movff   REG4,PREINC0
0004E4 C??? FFEC     01666                 movff   REG5,PREINC0
                     01667 ;       Save FSR2 on stack
0004E8 CFD9 FFEC     01668                 movff   FSR2L,PREINC0
0004EC CFDA FFEC     01669                 movff   FSR2H,PREINC0
                     01670 ;       if drive address is valid
0004F0 EE25 F000     01671                 lfsr    FSR2,uBuffer
0004F4 50DF          01672                 movf    INDF2,W
0004F6 6E??          01673                 movwf   REG0
0004F8 EC?? F???     01674                 call    IsAddrValid
0004FC E1??          01675                 bnz     CmdDiagUpldLpIf1f
                     01676 ;         Disable 1401 tape channel services
0004FE 6A??          01677                 clrf    ANLS_MMASK
000500 6A??          01678                 clrf    ANLS_EMASK
                     01679 ;         Convert the drive address to a bit mask
000502 50DF          01680                 movf    INDF2,W                 ;drive address
000504 EC?? F???     01681                 call    Addr2Mask               ;convert
                     01682 ;         Specify that the 1401 is in session with the specified drive
000508 52??          01683                 movf    ANLS_ACTIVE_DRV,F
                     01684 ;         Set the Tape Buffer and Event USB Buffer semaphore to Idle State.
00050A 6A??          01685                 clrf    TDS_SREQ
00050C 6A??          01686                 clrf    TDS_RCODE
00050E 6A??          01687                 clrf    EDS_SREQ
000510 6A??          01688                 clrf    EDS_RCODE
                     01689 ;         Get addressability to the specified drive's area
000512 50DF          01690                 movf    INDF2,W                 ;drive address
000514 EE?? F0??     01691                 lfsr    FSR2,drive1State-8      ;-8 for 0 origin NEW FSR2
000518 0B07          01692                 andlw   0x07
00051A 0D08          01693                 mullw   8                       ;drive address * 8
00051C 26D9          01694                 addwf   FSR2L                   ;add to FSR2 base addr
00051E 6AE8          01695                 clrf    WREG
000520 22DA          01696                 addwfc  FSR2H
                     01697 ;         Load the context of the addressed drive
000522 CFD9 F???     01698                 movff   FSR2L,ANLS_DRVCNTXT
000526 CFDA F???     01699                 movff   FSR2H,ANLS_DRVCNTXT+1
                     01700 ;         Create an Upload Request Event in the TISM's USB buffer
00052A CFDE F???     01701                 movff   POSTINC2,REG2   ;DSTATE
00052E CFDE F???     01702                 movff   POSTINC2,REG3   ;DS_LRC
000532 CFDE F???     01703                 movff   POSTINC2,REG4   ;DS_TAPEPOS low
000536 CFDE F???     01704                 movff   POSTINC2,REG5   ;DS_TAPEPOS high
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

00053A EE25 F080      01705                 lfsr    FSR2,eBuffer
00053E 0E83           01706                 movlw   usbEv_UpldReq
000540 6EDE           01707                 movwf   POSTINC2                ;event code
000542 C??? FFDE      01708                 movff   REG0,POSTINC2           ;drive address
000546 C??? FFDE      01709                 movff   REG2,POSTINC2   ;DSTATE
00054A C??? FFDE      01710                 movff   REG3,POSTINC2   ;DS_LRC
00054E C??? FFDE      01711                 movff   REG4,POSTINC2   ;DS_TAPEPOS low
000552 C??? FFDE      01712                 movff   REG5,POSTINC2   ;DS_TAPEPOS high
                      01713 ;         Set length of event message
000556 0E06           01714                 movlw   6
000558 6E??           01715                 movwf   EDS_COUNT
                      01716 ;         Store initial delay in its diagnostic state field
00055A EE25 F001      01717                 lfsr    FSR2,uBuffer+1
00055E CFDE F???      01718                 movff   POSTINC2,diagAlarmTime
000562 CFDE F???      01719                 movff   POSTINC2,diagAlarmTime+1
000566 CFDE F???      01720                 movff   POSTINC2,diagAlarmTime+2
00056A CFDE F???      01721                 movff   POSTINC2,diagAlarmTime+3
                      01722 ;         Store repeat delay in its diagnostic state field
00056E CFDE F???      01723                 movff   POSTINC2,diagDelayTime
000572 CFDE F???      01724                 movff   POSTINC2,diagDelayTime+1
000576 CFDE F???      01725                 movff   POSTINC2,diagDelayTime+2
00057A CFDE F???      01726                 movff   POSTINC2,diagDelayTime+3
                      01727 ;         Store the repeat count in in its analyzer state field
00057E CFDE F???      01728                 movff   POSTINC2,diagRepeatCnt
000582 CFDE F???      01729                 movff   POSTINC2,diagRepeatCnt+1
                      01730 ;         Enable the diagnostic timer alarm
000586 80??           01731                 bsf     diagCntl,diag_UpLdLb
                      01732 ;         Set response return code = RC_SUCCESS
000588 0E00           01733                 movlw   RC_SUCCESS
00058A 6F00           01734                 movwf   uBuffer
                      01735 ;       endif
00058C D???           01736                 bra     CmdDiagUpldLpIf1x
                      01737 ;       else drive address is not valid
                      01738 ;         Set response return code = RC_BAD_ADDR
00058E 0E01           01739 CmdDiagUpldLpIf1f movlw RC_BAD_ADDR
000590 6F00           01740                 movwf   uBuffer
                      01741 ;       endelse
000592                01742 CmdDiagUpldLpIf1x:
                      01743 ;       Set up event code (Now registers take on described purpose)
000592 0E04           01744                 movlw   usbCmd_DefVirtDrv
000594 52??           01745                 movf    REG0,F
                      01746 ;       Initialize the number of bytes to be transfered from the USB buffer = 1
000596 0E01           01747                 movlw   1
000598 52??           01748                 movf    REG1,F
                      01749 ;       Initialize the number of bytes to be transfered from the Tape Date buffer = 0
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

00059A 6A??           01750                 clrf    REG2
00059C 6A??           01751                 clrf    REG3
                      01752 ;       Send the message
00059E EC?? F???      01753                 call    SendUsbMsg
                      01754 ;Rtn:   Restore saved registers
0005A2 CFED FFDA      01755 CmdDiagUpldLpRtn movff  POSTDEC0,FSR2H
0005A6 CFED FFD9      01756                 movff   POSTDEC0,FSR2L
0005AA CFED F???      01757                 movff   POSTDEC0,REG5
0005AE CFED F???      01758                 movff   POSTDEC0,REG4
0005B2 CFED F???      01759                 movff   POSTDEC0,REG3
0005B6 CFED F???      01760                 movff   POSTDEC0,REG2
0005BA CFED F???      01761                 movff   POSTDEC0,REG1
0005BE CFED F???      01762                 movff   POSTDEC0,REG0
                      01763 ;       Restore the WREG
0005C2 50ED           01764                 movf    POSTDEC0,W
                      01765 ;       return
0005C4 0012           01766                 return
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     01767  PAGE
                     01768 ;*********************************************************************************
                     01769 ;       Send the "Truncated Message Event" message              pushes 7 bytes
                     01770 ;  void RespondTrunc()
                     01771 ;       Send the Truncated Event message. This message should include the data
                     01772 ;       that was received up to a maximum of 64 bytes. The message is composed
                     01773 ;       from data in the USB handler's state variables, data in the USB buffer,
                     01774 ;       and for cetrain commands, data in the Tape Data buffer.
                     01775 ;
                     01776 ;       REG usage:
                     01777 ;       REG0 = Command/Event code (by value input)
                     01778 ;       REG1 = number of bytes to be sent from USB xmit buffer (by value input)
                     01779 ;       REG2,3 = number of bytes to be sent from tape buffer (by value input)
                     01780 ;
                     01781 ;*********************************************************************************
0005C6               01782 RespondTrunc:
                     01783 ;       Save the WREG on the stack
0005C6 6EEC          01784                 movwf   PREINC0
                     01785 ;       Save REG0 through REG4 on stack
0005C8 C??? FFEC     01786                 movff   REG0,PREINC0
0005CC C??? FFEC     01787                 movff   REG1,PREINC0
0005D0 C??? FFEC     01788                 movff   REG2,PREINC0
0005D4 C??? FFEC     01789                 movff   REG3,PREINC0
                     01790 ;       Save FSR2 on stack
0005D8 CFD9 FFEC     01791                 movff   FSR2L,PREINC0
0005DC CFDA FFEC     01792                 movff   FSR2H,PREINC0
                     01793 ;       Set up event code
0005E0 0EFE          01794                 movlw   usbEv_TruncMsg
0005E2 52??          01795                 movf    REG0,F
                     01796 ;       Initialize the number of bytes to be transfered from the Tape Date buffer = 0
0005E4 6A??          01797                 clrf    REG2
0005E6 6A??          01798                 clrf    REG3
                     01799 ;       Set up byte count to be transmitted from the USB buffer, limit to 64 bytes
0005E8 50??          01800                 movf    usbCountL,W
0005EA 6E??          01801                 movwf   REG1
0005EC 0841          01802                 sublw   65
0005EE E6??          01803                 bn      RespondTruncLT65
0005F0 0E40          01804                 movlw   64
0005F2 52??          01805                 movf    REG1
0005F4               01806 RespondTruncLT65:
                     01807 ;       if data was placed in the Tape Data buffer
0005F4 AE??          01808                 btfss   usbStatus,usbSs_StTape
0005F6 D???          01809                 bra     RespondTruncIf1f
                     01810 ;         if command==usbCmd_DnldRecord && adj usbCount>2
0005F8 0E09          01811                 movlw   usbCmd_DnldRecord
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

0005FA 62??           01812                 cpfseq  usbState
0005FC D???           01813                 bra     RespondTruncIf2f
0005FE 50??           01814                 movf    REG1,W
000600 0803           01815                 sublw   3
000602 E7??           01816                 bnn     RespondTruncIf2f
                      01817 ;           Set tape data count = adjusted usb data count - 2
000604 50??           01818                 movf    REG1,W
000606 0802           01819                 sublw   2
000608 6E??           01820                 movwf   REG3
                      01821 ;           Set usb data count = 2
00060A 0E02           01822                 movlw   2
00060C 6E??           01823                 movwf   REG1
                      01824 ;         endif
00060E D???           01825                 bra     RespondTruncIf2x
                      01826 ;         elseif command==usbCmd_DiagUpldLp && usbCount>6
000610 0E10           01827 RespondTruncIf2f movlw  usbCmd_DiagUpldLp
000612 62??           01828                 cpfseq  usbState
000614 D???           01829                 bra     RespondTruncIf2x
000616 50??           01830                 movf    REG1,W
000618 0807           01831                 sublw   7
00061A E7??           01832                 bnn     RespondTruncIf2x
                      01833 ;           Set tape data count = adjusted usb data count - 6
00061C 50??           01834                 movf    REG1,W
00061E 0806           01835                 sublw   6
000620 6E??           01836                 movwf   REG3
                      01837 ;           Set usb data count = 6
000622 0E06           01838                 movlw   6
000624 6E??           01839                 movwf   REG1
                      01840 ;         endelseif
000626                01841 RespondTruncIf2x:
                      01842 ;       endif data was placed in the Tape Data buffer
000626                01843 RespondTruncIf1f:
                      01844 ;       Make room for 2 bytes at the front of the usb buffer by moving data forward
000626 C??? FFD9      01845                 movff   usbRecNxt,FSR2L         ;points to next avail loc
00062A C??? FFDA      01846                 movff   usbRecNxt+1,FSR2H
00062E 50DD           01847                 movf    POSTDEC2,W              ;back up pointer by 1 byte
000630 0E03           01848                 movlw   3
                      01849                         ;the below loop only works if the usb buffer size <=256 bytes
000632 CFDD FFDB      01850 RespondTruncL1s movff   POSTDEC2,PLUSW2         ;move data in buffer forward by 2 bytes
000636 66D9           01851                 tstfsz  FSR2L                   ;if low byte of FSR!=0
000638 D???           01852                 bra     RespondTruncL1s ; then keep looping
                      01853 ;       Insert the low order byte of the received length
00063A C??? FFDC      01854                 movff   usbLengthL,PREINC2
                      01855 ;       Insert the higgh order byte of the received length
00063E C??? FFDC      01856                 movff   usbLengthH,PREINC2
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     01857 ;       Send the message
000642 EC?? F???     01858                 call    SendUsbMsg
                     01859 ;Rtn:   Restore saved registers
000646 CFED FFDA     01860 RespondTruncRtn movff   POSTDEC0,FSR2H
00064A CFED FFD9     01861                 movff   POSTDEC0,FSR2L
00064E CFED F???     01862                 movff   POSTDEC0,REG3
000652 CFED F???     01863                 movff   POSTDEC0,REG2
000656 CFED F???     01864                 movff   POSTDEC0,REG1
00065A CFED F???     01865                 movff   POSTDEC0,REG0
                     01866 ;       Restore the WREG
00065E 50ED          01867                 movf    POSTDEC0,W
                     01868 ;       return
000660 0012          01869                 return
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     01870  PAGE
                     01871 ;*********************************************************************************
                     01872 ;       Send the "CRC Error Event" message                        pushes 9 bytes
                     01873 ;  void RespondCrcErr(REG0=expected CRC, REG1=received CRC)
                     01874 ;       Send the Truncated Event message. This message should include the data
                     01875 ;       that was received up to a maximum of 64 bytes. The message is composed
                     01876 ;       from data in the USB handler's state variables, data in the USB buffer,
                     01877 ;       and for cetrain commands, data in the Tape Data buffer.
                     01878 ;
                     01879 ;       REG usage:
                     01880 ;       REG0 = Command/Event code (by value input)
                     01881 ;       REG1 = number of bytes to be sent from USB xmit buffer (by value input)
                     01882 ;       REG2,3 = number of bytes to be sent from tape buffer (by value input)
                     01883 ;       REG4 = Expected CRC
                     01884 ;       REG5 = Received CRC
                     01885 ;
                     01886 ;*********************************************************************************
000662               01887 RespondCrcErr:
                     01888 ;       Save the WREG on the stack
000662 6EEC          01889                 movwf   PREINC0
                     01890 ;       Save REG0 through REG5 on stack
000664 C??? FFEC     01891                 movff   REG0,PREINC0
000668 C??? FFEC     01892                 movff   REG1,PREINC0
00066C C??? FFEC     01893                 movff   REG2,PREINC0
000670 C??? FFEC     01894                 movff   REG3,PREINC0
000674 C??? FFEC     01895                 movff   REG4,PREINC0
000678 C??? FFEC     01896                 movff   REG5,PREINC0
                     01897 ;       Save FSR2 on stack
00067C CFD9 FFEC     01898                 movff   FSR2L,PREINC0
000680 CFDA FFEC     01899                 movff   FSR2H,PREINC0
                     01900 ;       Move Expected CRC to REG4
000684 C??? F???     01901                 movff   REG0,REG4
                     01902 ;       Move Received CRC to REG5
000688 C??? F???     01903                 movff   REG1,REG5
                     01904 ;       Set up event code
00068C 0EFD          01905                 movlw   usbEv_CRCError
00068E 52??          01906                 movf    REG0,F
                     01907 ;       Initialize the number of bytes to be transfered from the Tape Date buffer = 0
000690 6A??          01908                 clrf    REG2
000692 6A??          01909                 clrf    REG3
                     01910 ;       Set up byte count to be transmitted from the USB buffer, limit to 64 bytes
000694 50??          01911                 movf    usbCountL,W
000696 6E??          01912                 movwf   REG1
000698 0841          01913                 sublw   65
00069A E6??          01914                 bn      RespondCrcErrLT65
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

00069C 0E40           01915                 movlw   64
00069E 52??           01916                 movf    REG1
0006A0                01917 RespondCrcErrLT65:
                      01918 ;       if data was placed in the Tape Data buffer
0006A0 AE??           01919                 btfss   usbStatus,usbSs_StTape
0006A2 D???           01920                 bra     RespondCrcErrIf1f
                      01921 ;         if command==usbCmd_DnldRecord && adj usbCount>2
0006A4 0E09           01922                 movlw   usbCmd_DnldRecord
0006A6 62??           01923                 cpfseq  usbState
0006A8 D???           01924                 bra     RespondCrcErrIf2f
0006AA 50??           01925                 movf    REG1,W
0006AC 0803           01926                 sublw   3
0006AE E7??           01927                 bnn     RespondCrcErrIf2f
                      01928 ;           Set tape data count = adjusted usb data count - 2
0006B0 50??           01929                 movf    REG1,W
0006B2 0802           01930                 sublw   2
0006B4 6E??           01931                 movwf   REG3
                      01932 ;           Set usb data count = 2
0006B6 0E02           01933                 movlw   2
0006B8 6E??           01934                 movwf   REG1
                      01935 ;         endif
0006BA D???           01936                 bra     RespondCrcErrIf2x
                      01937 ;         elseif command==usbCmd_DiagUpldLp && usbCount>6
0006BC 0E10           01938 RespondCrcErrIf2f movlw usbCmd_DiagUpldLp
0006BE 62??           01939                 cpfseq  usbState
0006C0 D???           01940                 bra     RespondCrcErrIf2x
0006C2 50??           01941                 movf    REG1,W
0006C4 0807           01942                 sublw   7
0006C6 E7??           01943                 bnn     RespondCrcErrIf2x
                      01944 ;           Set tape data count = adjusted usb data count - 6
0006C8 50??           01945                 movf    REG1,W
0006CA 0806           01946                 sublw   6
0006CC 6E??           01947                 movwf   REG3
                      01948 ;           Set usb data count = 6
0006CE 0E06           01949                 movlw   6
0006D0 6E??           01950                 movwf   REG1
                      01951 ;         endelseif
0006D2                01952 RespondCrcErrIf2x:
                      01953 ;       endif data was placed in the Tape Data buffer
0006D2                01954 RespondCrcErrIf1f:
                      01955 ;       Make room for 2 bytes at the front of the usb buffer by moving data forward
0006D2 C??? FFD9      01956                 movff   usbRecNxt,FSR2L         ;points to next avail loc
0006D6 C??? FFDA      01957                 movff   usbRecNxt+1,FSR2H
0006DA 50DD           01958                 movf    POSTDEC2,W              ;back up pointer by 1 byte
0006DC 0E03           01959                 movlw   3
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     01960                          ;the below loop only works if the usb buffer size <=256 bytes
0006DE CFDD FFDB     01961 RespondCrcErrL1s movff  POSTDEC2,PLUSW2         ;move data in buffer forward by 2 bytes
0006E2 66D9          01962                 tstfsz  FSR2L                  ;if low byte of FSR!=0
0006E4 D???          01963                 bra     RespondCrcErrL1s       ; then keep looping
                     01964 ;       Insert the received CRC
0006E6 C??? FFDC     01965                 movff   REG5,PREINC2
                     01966 ;       Insert the expected crc
0006EA C??? FFDC     01967                 movff   REG4,PREINC2
                     01968 ;       Send the message
0006EE EC?? F???     01969                 call    SendUsbMsg
                     01970 ;Rtn:   Restore saved registers
0006F2 CFED FFDA     01971 RespondCrcErrRtn movff  POSTDEC0,FSR2H
0006F6 CFED FFD9     01972                 movff   POSTDEC0,FSR2L
0006FA CFED F???     01973                 movff   POSTDEC0,REG5
0006FE CFED F???     01974                 movff   POSTDEC0,REG4
000702 CFED F???     01975                 movff   POSTDEC0,REG3
000706 CFED F???     01976                 movff   POSTDEC0,REG2
00070A CFED F???     01977                 movff   POSTDEC0,REG1
00070E CFED F???     01978                 movff   POSTDEC0,REG0
                     01979 ;       Restore the WREG
000712 50ED          01980                 movf    POSTDEC0,W
                     01981 ;       return
000714 0012          01982                 return
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     01983  PAGE
                     01984 ;*************************************************************************
                     01985 ;       Send the "Invalid Command Event" message               pushes 5 bytes
                     01986 ;  void RespondInvCmd()
                     01987 ;       Send the Invalid Command Event message. This message should include the
                     01988 ;       data that was received up to a maximum of 64 bytes. The message is
                     01989 ;       composed from data in the USB handler's state variables, data in the USB
                     01990 ;       buffer, and for cetrain commands, data in the Tape Data buffer.
                     01991 ;
                     01992 ;       REG usage:
                     01993 ;       REG0 = Command/Event code (by value input)
                     01994 ;       REG1 = number of bytes to be sent from USB xmit buffer (by value input)
                     01995 ;       REG2,3 = number of bytes to be sent from tape buffer (by value input)
                     01996 ;
                     01997 ;*************************************************************************
000716               01998 RespondInvCmd:
                     01999 ;       Save the WREG on the stack
000716 6EEC          02000                 movwf   PREINC0
                     02001 ;       Save REG0 through REG3 on stack
000718 C??? FFEC     02002                 movff   REG0,PREINC0
00071C C??? FFEC     02003                 movff   REG1,PREINC0
000720 C??? FFEC     02004                 movff   REG2,PREINC0
000724 C??? FFEC     02005                 movff   REG3,PREINC0
                     02006 ;       Set up event code
000728 0EFC          02007                 movlw   usbEv_InvalidCmd
00072A 52??          02008                 movf    REG0,F
                     02009 ;       Initialize the number of bytes to be transfered from the Tape Date buffer = 0
00072C 6A??          02010                 clrf    REG2
00072E 6A??          02011                 clrf    REG3
                     02012 ;       Set up byte count to be transmitted from the USB buffer, limit to 64 bytes
000730 50??          02013                 movf    usbCountL,W
000732 6E??          02014                 movwf   REG1
000734 0841          02015                 sublw   65
000736 E6??          02016                 bn      RespondInvCmdLT65
000738 0E40          02017                 movlw   64
00073A 52??          02018                 movf    REG1
00073C               02019 RespondInvCmdLT65:
                     02020 ;       if data was placed in the Tape Data buffer
00073C AE??          02021                 btfss   usbStatus,usbSs_StTape
00073E D???          02022                 bra     RespondInvCmdIf1f
                     02023 ;         if command==usbCmd_DnldRecord && adj usbCount>2
000740 0E09          02024                 movlw   usbCmd_DnldRecord
000742 62??          02025                 cpfseq  usbState
000744 D???          02026                 bra     RespondInvCmdIf2f
000746 50??          02027                 movf    REG1,W
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

000748 0803           02028                 sublw   3
00074A E7??           02029                 bnn     RespondInvCmdIf2f
                      02030 ;           Set tape data count = adjusted usb data count - 2
00074C 50??           02031                 movf    REG1,W
00074E 0802           02032                 sublw   2
000750 6E??           02033                 movwf   REG3
                      02034 ;           Set usb data count = 2
000752 0E02           02035                 movlw   2
000754 6E??           02036                 movwf   REG1
                      02037 ;         endif
000756 D???           02038                 bra     RespondInvCmdIf2x
                      02039 ;         elseif command==usbCmd_DiagUpldLp && usbCount>6
000758 0E10           02040 RespondInvCmdIf2f movlw usbCmd_DiagUpldLp
00075A 62??           02041                 cpfseq  usbState
00075C D???           02042                 bra     RespondInvCmdIf2x
00075E 50??           02043                 movf    REG1,W
000760 0807           02044                 sublw   7
000762 E7??           02045                 bnn     RespondInvCmdIf2x
                      02046 ;           Set tape data count = adjusted usb data count - 6
000764 50??           02047                 movf    REG1,W
000766 0806           02048                 sublw   6
000768 6E??           02049                 movwf   REG3
                      02050 ;           Set usb data count = 6
00076A 0E06           02051                 movlw   6
00076C 6E??           02052                 movwf   REG1
                      02053 ;         endelseif
00076E                02054 RespondInvCmdIf2x:
                      02055 ;       endif data was placed in the Tape Data buffer
00076E                02056 RespondInvCmdIf1f:
                      02057 ;       Send the message
00076E EC?? F???      02058                 call    SendUsbMsg
                      02059 ;Rtn:   Restore saved registers
000772 CFED F???      02060 RespondInvCmdRtn        movff   POSTDEC0,REG3
000776 CFED F???      02061                 movff   POSTDEC0,REG2
00077A CFED F???      02062                 movff   POSTDEC0,REG1
00077E CFED F???      02063                 movff   POSTDEC0,REG0
                      02064 ;       Restore the WREG
000782 50ED           02065                 movf    POSTDEC0,W
                      02066 ;       return
000784 0012           02067                 return
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     02068  PAGE
                     02069 ;*************************************************************************
                     02070 ;       Send USB Message                                        pushes 11 bytes
                     02071 ; void  SendUsbMsg(REG0=cmd/event, REG1=bytes_in_USB_buffer,
                     02072 ;                   REG2=bytes_in_tape_buffer_low,3=bytes_in_tape_buffer_high)
                     02073 ;       A message constructed from the input parameters and transmitted on the
                     02074 ;       usb bus. If data to be sent exists in both the USB and tape buffers, the
                     02075 ;       data from the usb buffer is sent first.
                     02076 ;       If sending the message times out, then the analyzer status is updated
                     02077 ;       to show the failure.
                     02078 ;
                     02079 ;       REG usage:
                     02080 ;       REG0 = Command/Event code (by value input)
                     02081 ;       REG1 = number of bytes to be sent from USB xmit buffer (by value input)
                     02082 ;       REG2,3 = number of bytes to be sent from tape buffer (by value input)
                     02083 ;       REG4,5 = computed total message length
                     02084 ;       REG6 = Accumulated CRC
                     02085 ;       REG7 = not used.
                     02086 ;
                     02087 ;       FSR2 is used as a buffer pointer
                     02088 ;
                     02089 ;*************************************************************************
000786               02090 SendUsbMsg:
                     02091 ;       Save the WREG on the stack
000786 6EEC          02092                 movwf   PREINC0
                     02093 ;       Save REG0 through REG7 on stack
000788 C??? FFEC     02094                 movff   REG0,PREINC0
00078C C??? FFEC     02095                 movff   REG1,PREINC0
000790 C??? FFEC     02096                 movff   REG2,PREINC0
000794 C??? FFEC     02097                 movff   REG3,PREINC0
000798 C??? FFEC     02098                 movff   REG4,PREINC0
00079C C??? FFEC     02099                 movff   REG5,PREINC0
0007A0 C??? FFEC     02100                 movff   REG6,PREINC0
0007A4 C??? FFEC     02101                 movff   REG7,PREINC0
                     02102 ;       Save FRS2 on stack
0007A8 CFD9 FFEC     02103                 movff   FSR2L,PREINC0
0007AC CFDA FFEC     02104                 movff   FSR2H,PREINC0
                     02105 ;       Try
                     02106 ;         Compute message_length = length(usb buffer data) + length(tape buffer data) + 1
0007B0 C??? F???     02107                 movff   REG2,REG4       ;compute length in REG4,5
0007B4 C??? F???     02108                 movff   REG3,REG5
0007B8 80D8          02109                 bsf     STATUS,C        ;add 1 for command/event code byte
0007BA 50??          02110                 movf    REG1,W
0007BC 22??          02111                 addwfc  REG4,F
0007BE 6AE8          02112                 clrf    WREG
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

0007C0 22??           02113                addwfc  REG5,F
                      02114 ;         Initialize working CRC
0007C2 6A??           02115                clrf    REG6
                      02116 ;         Send Start flag
0007C4 0EF0           02117                movlw   USB_START_FLAG
0007C6 EC?? F???      02118                call    SendUsbChar
0007CA E2??           02119                bc      SendUsbMsgCatch
                      02120 ;         Send Length bytes
0007CC 50??           02121                movf    REG4,W
0007CE EC?? F???      02122                call    SendUsbChar
0007D2 E2??           02123                bc      SendUsbMsgCatch
0007D4 50??           02124                movf    REG5,W
0007D6 EC?? F???      02125                call    SendUsbChar
0007DA E2??           02126                bc      SendUsbMsgCatch
                      02127 ;         Send Response/Event Code and accumulate CRC
0007DC 50??           02128                movf    REG0,W
0007DE 26??           02129                addwf   REG6,F          ;accum lrc
0007E0 EC?? F???      02130                call    SendUsbChar
0007E4 E2??           02131                bc      SendUsbMsgCatch
                      02132 ;         Send Timestamp and accumulate CRC
0007E6 EC?? F???      02133                call    GetTimestamp    ;timestamp is on stack
0007EA 50ED           02134                movf    POSTDEC0,W      ;get timestamp byte0
0007EC 26??           02135                addwf   REG6,F          ;accum lrc
0007EE EC?? F???      02136                call    SendUsbChar
0007F2 E2??           02137                bc      SendUsbMsgCatch
0007F4 50ED           02138                movf    POSTDEC0,W      ;get timestamp byte1
0007F6 26??           02139                addwf   REG6,F          ;accum lrc
0007F8 EC?? F???      02140                call    SendUsbChar
0007FC E2??           02141                bc      SendUsbMsgCatch
0007FE 50ED           02142                movf    POSTDEC0,W      ;get timestamp byte2
000800 26??           02143                addwf   REG6,F          ;accum lrc
000802 EC?? F???      02144                call    SendUsbChar
000806 E2??           02145                bc      SendUsbMsgCatch
000808 50ED           02146                movf    POSTDEC0,W      ;get timestamp byte3
00080A 26??           02147                addwf   REG6,F          ;accum lrc
00080C EC?? F???      02148                call    SendUsbChar
000810 E2??           02149                bc      SendUsbMsgCatch
                      02150 ;         while (bytes_in_USB_buffer==0)
000812 EE25 F000      02151                lfsr    FSR2,uBuffer
000816 50??           02152 SendUsbMsgL1s  movf    REG1,W
000818 E0??           02153                bz      SendUsbMsgL1x
                      02154 ;           Send next character from usb buffer and accumulate CRC
00081A 50DE           02155                movf    POSTINC2,W
00081C 26??           02156                addwf   REG6,F          ;accum lrc
00081E EC?? F???      02157                call    SendUsbChar
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

000822 E2??           02158                 bc      SendUsbMsgCatch
                      02159 ;           bytes_in_USB_buffer--
000824 0E01           02160                 movlw   1
000826 5E??           02161                 subwf   REG1,F
                      02162 ;         endwhile
000828 D???           02163                 bra     SendUsbMsgL1s
00082A                02164 SendUsbMsgL1x:
                      02165 ;         while (bytes_in_tape_buffer!=0)
00082A EE26 F000      02166                 lfsr    FSR2,tBuffer
00082E 50??           02167 SendUsbMsgL2s   movf    REG2,W
000830 E1??           02168                 bnz     SendUsbMsgL2ss
000832 50??           02169                 movf    REG3,W
000834 E0??           02170                 bz      SendUsbMsgL2x
000836                02171 SendUsbMsgL2ss:
                      02172 ;           Send next character from tape data buffer and accumulate CRC
000836 50DE           02173                 movf    POSTINC2,W
000838 26??           02174                 addwf   REG6,F          ;accum lrc
00083A EC?? F???      02175                 call    SendUsbChar
00083E E2??           02176                 bc      SendUsbMsgCatch
                      02177 ;           bytes_in_tape_buffer--
000840 0E01           02178                 movlw   1
000842 5E??           02179                 subwf   REG2,F
000844 6AE8           02180                 clrf    WREG
000846 5A??           02181                 subwfb  REG3,F
                      02182 ;         endwhile
000848 D???           02183                 bra     SendUsbMsgL2s
00084A                02184 SendUsbMsgL2x:
                      02185 ;         Send CRC
00084A 50??           02186                 movf    REG6,W
00084C 6CE8           02187                 negf    WREG            ;form lrc
00084E EC?? F???      02188                 call    SendUsbChar
000852 E2??           02189                 bc      SendUsbMsgCatch
                      02190 ;         Send Stop flag
000854 0EF0           02191                 movlw   USB_START_FLAG
000856 EC?? F???      02192                 call    SendUsbChar
00085A E2??           02193                 bc      SendUsbMsgCatch
                      02194 ;       endtry
00085C D???           02195                 bra     SendUsbMsgRtn
                      02196 ;       catch Condition(USB timeout)
00085E                02197 SendUsbMsgCatch:
                      02198 ;         Indicate timeout in analyzer status
00085E 86??           02199                 bsf     ANLS_STATUS,ANLS_UsbTimeout
                      02200 ;       endcatch
                      02201 ;Rtn:   Restore saved registers, instructions from here down can't change STATUS_C
000860 CFED FFDA      02202 SendUsbMsgRtn   movff   POSTDEC0,FSR2H
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

000864 CFED FFD9      02203                movff   POSTDEC0,FSR2L
000868 CFED F???      02204                movff   POSTDEC0,REG7
00086C CFED F???      02205                movff   POSTDEC0,REG6
000870 CFED F???      02206                movff   POSTDEC0,REG5
000874 CFED F???      02207                movff   POSTDEC0,REG4
000878 CFED F???      02208                movff   POSTDEC0,REG3
00087C CFED F???      02209                movff   POSTDEC0,REG2
000880 CFED F???      02210                movff   POSTDEC0,REG1
000884 CFED F???      02211                movff   POSTDEC0,REG0
                      02212 ;       Restore the WREG
000888 50ED           02213                movf    POSTDEC0,W
                      02214 ;       return
00088A 0012           02215                return
                      02216
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     02217  PAGE
                     02218 ;*******************************************************************************
                     02219 ;       Transmit Event on USB                                          pushes 11 bytes
                     02220 ; void  XmitEvent()
                     02221 ;       A message constructed from Event Buffer and its byte count
                     02222 ;       usb bus.
                     02223 ;       If sending the message times out, then the analyzer status is updated
                     02224 ;       to show the failure.
                     02225 ;
                     02226 ;       REG usage:
                     02227 ;       REG1 = number of bytes to be sent from USB xmit buffer (decrimented)
                     02228 ;       REG6 = Accumulated CRC
                     02229 ;
                     02230 ;       FSR2 is used as a buffer pointer
                     02231 ;
                     02232 ;*******************************************************************************
00088C               02233 XmitEvent:
                     02234 ;       Save the WREG on the stack
00088C 6EEC          02235                 movwf   PREINC0
                     02236 ;       Save REG1 and REG6 on stack
00088E C??? FFEC     02237                 movff   REG1,PREINC0
000892 C??? FFEC     02238                 movff   REG6,PREINC0
                     02239 ;       Save FRS2 on stack
000896 CFD9 FFEC     02240                 movff   FSR2L,PREINC0
00089A CFDA FFEC     02241                 movff   FSR2H,PREINC0
                     02242 ;       Try
                     02243 ;         Initialize working CRC
00089E 6A??          02244                 clrf    REG6
                     02245 ;         Send Start flag
0008A0 0EF0          02246                 movlw   USB_START_FLAG
0008A2 EC?? F???     02247                 call    SendUsbChar
0008A6 E2??          02248                 bc      XmitEventCatch
                     02249 ;         Send Length bytes
0008A8 50??          02250                 movf    EDS_COUNT,W
0008AA 6E??          02251                 movwf   REG1
0008AC EC?? F???     02252                 call    SendUsbChar
0008B0 E2??          02253                 bc      XmitEventCatch
0008B2 6AE8          02254                 clrf    WREG
0008B4 EC?? F???     02255                 call    SendUsbChar
0008B8 E2??          02256                 bc      XmitEventCatch
                     02257 ;         Event Code and accumulate CRC
0008BA EE25 F080     02258                 lfsr    FSR2,eBuffer
0008BE 50DE          02259                 movf    POSTINC2,W
0008C0 26??          02260                 addwf   REG6,F          ;accum lrc
0008C2 EC?? F???     02261                 call    SendUsbChar
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

0008C6 E2??           02262                 bc      XmitEventCatch
0008C8 06??           02263                 decf    REG1
                      02264 ;         Send Timestamp and accumulate CRC
0008CA EC?? F???      02265                 call    GetTimestamp    ;timestamp is on stack
0008CE 50ED           02266                 movf    POSTDEC0,W      ;get timestamp byte0
0008D0 26??           02267                 addwf   REG6,F          ;accum lrc
0008D2 EC?? F???      02268                 call    SendUsbChar
0008D6 E2??           02269                 bc      XmitEventCatch
0008D8 50ED           02270                 movf    POSTDEC0,W      ;get timestamp byte1
0008DA 26??           02271                 addwf   REG6,F          ;accum lrc
0008DC EC?? F???      02272                 call    SendUsbChar
0008E0 E2??           02273                 bc      XmitEventCatch
0008E2 50ED           02274                 movf    POSTDEC0,W      ;get timestamp byte2
0008E4 26??           02275                 addwf   REG6,F          ;accum lrc
0008E6 EC?? F???      02276                 call    SendUsbChar
0008EA E2??           02277                 bc      XmitEventCatch
0008EC 50ED           02278                 movf    POSTDEC0,W      ;get timestamp byte3
0008EE 26??           02279                 addwf   REG6,F          ;accum lrc
0008F0 EC?? F???      02280                 call    SendUsbChar
0008F4 E2??           02281                 bc      XmitEventCatch
                      02282 ;         while (bytes_in_Event_buffer==0)
0008F6 50??           02283 XmitEventL1s    movf    REG1,W
0008F8 E0??           02284                 bz      XmitEventL1x
                      02285 ;           Send next character from event buffer and accumulate CRC
0008FA 50DE           02286                 movf    POSTINC2,W
0008FC 26??           02287                 addwf   REG6,F          ;accum lrc
0008FE EC?? F???      02288                 call    SendUsbChar
000902 E2??           02289                 bc      XmitEventCatch
                      02290 ;           bytes_in_event_buffer--
000904 06??           02291                 decf    REG1
                      02292 ;         endwhile
000906 D???           02293                 bra     XmitEventL1s
000908                02294 XmitEventL1x:
                      02295 ;         Send CRC
000908 50??           02296                 movf    REG6,W
00090A 6CE8           02297                 negf    WREG            ;form lrc
00090C EC?? F???      02298                 call    SendUsbChar
000910 E2??           02299                 bc      XmitEventCatch
                      02300 ;         Send Stop flag
000912 0EF0           02301                 movlw   USB_START_FLAG
000914 EC?? F???      02302                 call    SendUsbChar
000918 E2??           02303                 bc      XmitEventCatch
                      02304 ;       endtry
00091A D???           02305                 bra     XmitEventRtn
                      02306 ;       catch Condition(USB timeout)
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

00091C                02307 XmitEventCatch:
                      02308 ;         Indicate timeout in analyzer status
00091C 86??           02309                 bsf     ANLS_STATUS,ANLS_UsbTimeout
                      02310 ;       endcatch
                      02311 ;Rtn:   Restore saved registers, instructions from here down can't change STATUS_C
00091E CFED FFDA      02312 XmitEventRtn    movff   POSTDEC0,FSR2H
000922 CFED FFD9      02313                 movff   POSTDEC0,FSR2L
000926 CFED F???      02314                 movff   POSTDEC0,REG6
00092A CFED F???      02315                 movff   POSTDEC0,REG1
                      02316 ;       Restore the WREG
00092E 50ED           02317                 movf    POSTDEC0,W
                      02318 ;       return
000930 0012           02319                 return
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     02320  PAGE
                     02321 ;*********************************************************************
                     02322 ;       Send character to USB interface                          pushes 9 bytes
                     02323 ; (STATUS_C timeout) SendUsbChar(WREG=character to be sent)
                     02324 ;       Notes the time when the USB interface started blocking execution.
                     02325 ;       If it blocks for more than xxx seconds, transmission is aborted.
                     02326 ;
                     02327 ;       Stack setup
                     02328 ;       0 to -3  = Sampled Timestamp, pushed by GetTimestamp
                     02329 ;         -4   = Saved REG7
                     02330 ;         -5   = Saved REG6
                     02331 ;         -6   = Saved REG5
                     02332 ;         -7   = Saved REG4
                     02333 ;         -8   = Saved REG3
                     02334 ;         -9   = Saved REG2
                     02335 ;        -10   = Saved REG1
                     02336 ;        -11   = Saved REG0
                     02337 ;        -12   = Saved WREG
                     02338 ;
                     02339 ;       REG0 through REG3 are used to hold an int alarm value.
                     02340 ;       REG4 through REG7 are used for time stamp arithmatic
                     02341 ;*********************************************************************
000932               02342 SendUsbChar:
                     02343 ;       Save the character to be sent (WREG) on the stack
000932 6EEC          02344                 movwf   PREINC0
                     02345 ;       Save REG0 through REG7 on stack
000934 C??? FFEC     02346                 movff   REG0,PREINC0
000938 C??? FFEC     02347                 movff   REG1,PREINC0
00093C C??? FFEC     02348                 movff   REG2,PREINC0
000940 C??? FFEC     02349                 movff   REG3,PREINC0
000944 C??? FFEC     02350                 movff   REG4,PREINC0
000948 C??? FFEC     02351                 movff   REG5,PREINC0
00094C C??? FFEC     02352                 movff   REG6,PREINC0
000950 C??? FFEC     02353                 movff   REG7,PREINC0
                     02354 ;       Set Alarm to OFF
000954 6A??          02355                 clrf    REG0            ;lowest order byte
000956 6A??          02356                 clrf    REG1
000958 6A??          02357                 clrf    REG2
00095A 6A??          02358                 clrf    REG3
                     02359 ;       while the USB chip is not ready to accept a character
00095C               02360 SendUsbCharLp:
                     02361             if is_simulating
                     02362                 movf    REG0,W          ;if simulating, proceed when alarm low byte==0
                     02363                 bz      SendUsbCharEL
                     02364             else ;not simulating
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     02365                 input   TXE                     ;get TXE#, put in carry bit
00095C 90D8              M                 bcf     STATUS,C
00095E B881              M                 btfsc   (31756)>>3&0xff,(31756)&0x07,0
000960 80D8              M                 bsf     STATUS,C
000962 E3??          02366                 bnc     SendUsbCharEL           ; carry bit off means room is available
                     02367               endif ;simulating test
                     02368 ;         if there is no alarm time set
000964 66??          02369                 tstfsz  REG0                    ;test lowest order alarm byte
000966 D???          02370                 bra     SendUsbCharNSet
000968 66??          02371                 tstfsz  REG1
00096A D???          02372                 bra     SendUsbCharNSet
00096C 66??          02373                 tstfsz  REG2
00096E D???          02374                 bra     SendUsbCharNSet
000970 66??          02375                 tstfsz  REG3                    ;test highest order alarm byte
000972 D???          02376                 bra     SendUsbCharNSet
                     02377 ;           Set an alarm time about 1 minute in the future
  00047869           02378 ONE_MINUTE      EQU     292969  ;ticks,  each tick is .2 milliseconds
000974 EC?? F???     02379 SendUsbCharNSet call    GetTimestamp
000978 CFED F???     02380                 movff   POSTDEC0,REG0           ;pop Timestamp into REG0-REG3
00097C CFED F???     02381                 movff   POSTDEC0,REG1
000980 CFED F???     02382                 movff   POSTDEC0,REG2
000984 CFED F???     02383                 movff   POSTDEC0,REG3
000988 0E01          02384                 movlw   ONE_MINUTE&&0xff        ;advance timestamp one minute
00098A 26??          02385                 addwf   REG0,F                  ;form byte 0
00098C 0E01          02386                 movlw   ONE_MINUTE>>8&&0xff
00098E 22??          02387                 addwfc  REG1,F                  ;form byte 1
000990 0E01          02388                 movlw   ONE_MINUTE>>16&&0xff
000992 22??          02389                 addwfc  REG2,F                  ;form byte 2
000994 0E00          02390                 movlw   ONE_MINUTE>>24&&0xff
000996 22??          02391                 addwfc  REG3,F                  ;form byte 3
                     02392 ;         endif
000998 D???          02393                 bra SendUsbCharLG       ;continue
                     02394 ;         elseif the alarm time has been exceeded
00099A EC?? F???     02395 SendUsbCharSet  call    GetTimestamp
00099E CFED F???     02396                 movff   POSTDEC0,REG4           ;pop Timestamp into REG4-REG7
0009A2 CFED F???     02397                 movff   POSTDEC0,REG5
0009A6 CFED F???     02398                 movff   POSTDEC0,REG6
0009AA CFED F???     02399                 movff   POSTDEC0,REG7
0009AE 50??          02400                 movf    REG0,W
0009B0 5E??          02401                 subwf   REG4,F                  ;current_time - alarm_time
0009B2 50??          02402                 movf    REG1,W
0009B4 5A??          02403                 subwfb  REG5,F
0009B6 50??          02404                 movf    REG2,W
0009B8 5A??          02405                 subwfb  REG6,F
0009BA 50??          02406                 movf    REG3,W
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

0009BC 5A??           02407                 subwfb  REG7,F
0009BE E6??           02408                 bn      SendUsbCharLG           ;neg means no timeout yet
                      02409 ;       Indicate timout occurred
0009C0 80D8           02410                 bsf     STATUS,C        ;Instructions after this cant change STATUS_C
                      02411 ;           goto Rtn
0009C2 D???           02412                 bra     SendUsbCharRtn
                      02413 ;         endelseif
                      02414 ;       endewhile USB chip is not ready to accept a character
0009C4 D???           02415 SendUsbCharLG:  bra     SendUsbCharLp   ;goto the top of the while loop
0009C6                02416 SendUsbCharEL:
                      02417 ;       Give it the character
0009C6 0E00           02418                 movlw   SELUSB
0009C8 6E8D           02419                 movwf   LATE            ;select the USB device
0009CA 0E02           02420                 movlw   2
0009CC CFEB FF8C      02421                 movff   PLUSW0,LATD     ;write the data to the data bus
                      02422                 output_high USB_WR      ;strobe the USB chip to accept it
0009D0 8486               M                 bsf     (31794)>>3&0xff,(31794)&0x07,0
                      02423                 output_low  USB_WR      ;writes on the high to low transition
0009D2 9486               M                 bcf     (31794)>>3&0xff,(31794)&0x07,0
                      02424 ;       Indicate good completion
0009D4 90D8           02425                 bcf     STATUS,C
                      02426 ;Rtn:   Restore saved registers, instructions from here down can't change STATUS_C
0009D6 CFED F???      02427 SendUsbCharRtn  movff   POSTDEC0,REG7
0009DA CFED F???      02428                 movff   POSTDEC0,REG6
0009DE CFED F???      02429                 movff   POSTDEC0,REG5
0009E2 CFED F???      02430                 movff   POSTDEC0,REG4
0009E6 CFED F???      02431                 movff   POSTDEC0,REG3
0009EA CFED F???      02432                 movff   POSTDEC0,REG2
0009EE CFED F???      02433                 movff   POSTDEC0,REG1
0009F2 CFED F???      02434                 movff   POSTDEC0,REG0
                      02435 ;       Restore the WREG
0009F6 50ED           02436                 movf    POSTDEC0,W
                      02437 ;       return
0009F8 0012           02438                 return
                      02439         SUBTITLE "Utility Subroutines"
```

```
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     02440  PAGE
                     02441 ;*******************************************************************************
                     02442 ; Drive address to bit conversion subroutine"                  pushes 3 bytes
                     02443 ;       (WREG=bit mask) Addr2Mask(WREG=address)
                     02444 ;
0009FA               02445 Addr2Mask:
                     02446 ;       Save Table Reg on stack
0009FA CFF6 FFEC     02447                 movff   TBLPTRL,PREINC0
0009FE CFF7 FFEC     02448                 movff   TBLPTRH,PREINC0
000A02 CFF8 FFEC     02449                 movff   TBLPTRU,PREINC0
                     02450 ;       Push addr offset into table pointer reg
000A06 52F6          02451                 movf    TBLPTRL,F
000A08 6AF7          02452                 clrf    TBLPTRH
000A0A 6AF8          02453                 clrf    TBLPTRU
                     02454 ;       Add table base address into table pointer reg
000A0C 0E??          02455                 movlw   low Addr2MaskTbl
000A0E 26F6          02456                 addwf   TBLPTRL
000A10 0E??          02457                 movlw   high Addr2MaskTbl
000A12 22F7          02458                 addwfc  TBLPTRH
000A14 0E??          02459                 movlw   upper Addr2MaskTbl
000A16 22F8          02460                 addwfc  TBLPTRU
                     02461 ;       Load the byte from the table
000A18 0008          02462                 tblrd   *
                     02463 ;Rtn:   Restore saved registers
000A1A CFED FFF8     02464 Addr2MaskRtn    movff   POSTDEC0,TBLPTRU
000A1E CFED FFF7     02465                 movff   POSTDEC0,TBLPTRH
000A22 CFED FFF6     02466                 movff   POSTDEC0,TBLPTRL
                     02467 ;       return with result in WREG
000A26 0012          02468                 return
000A28 0201 0804 2010 02469 Addr2MaskTbl    db      0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80
       8040
```

LOC OBJECT CODE   LINE SOURCE TEXT
  VALUE

```
                     02470  PAGE
                     02471 ;*******************************************************************************
                     02472 ; Drive address to bit conversion subroutine"                   pushes 1 bytes
                     02473 ;       (WREG=(0==True)) IsAddrValid(WREG=address)
                     02474 ;
                     02475 ;       Check if WREG is between 1 and 6.
                     02476 ;       If it is, then return 0, else return RC_BAD_ADDR.
                     02477 ;
000A30               02478 IsAddrValid:
                     02479 ;       Push WREG onto stack
000A30 CFE8 FFEC     02480                 movff   WREG,PREINC0
                     02481 ;       if 1<= address <=6
000A34 6AE8          02482                 clrf    WREG
000A36 64DF          02483                 cpfsgt  INDF2
000A38 D???          02484                 bra     IsAddrValidIf1f
000A3A 0E07          02485                 movlw   7
000A3C 60DF          02486                 cpfslt  INDF2
000A3E D???          02487                 bra     IsAddrValidIf1f
                     02488 ;         Set saved WREG to 0
000A40 6ADF          02489                 clrf    INDF2
                     02490 ;       endif
000A42 D???          02491                 bra     IsAddrValidRtn
                     02492 ;       else not (1<= address <=6)
000A44               02493 IsAddrValidIf1f:
                     02494 ;       Set saved WREG to 1
000A44 0E01          02495                 movlw   RC_BAD_ADDR
000A46 52DF          02496                 movf    INDF2,F
                     02497 ;Rtn    Restore the WREG
000A48 CFED FFE8     02498 IsAddrValidRtn  movff   POSTDEC0,WREG
                     02499 ;       return
000A4C 0012          02500                 return
                     02501
                     02502 ;End of program
                     02503 ;*******************************************************************************
```

LOC OBJECT CODE   LINE SOURCE TEXT
  VALUE

```
                     02504  PAGE
                     02505                  SUBTITLE "Test data from the USB port"
                     02506 UsbFakeB         org     0x8000  ;for multibyte integers, low order byte comes first
                     02507 ;       Define drive 1
008000 0BF0 0000     02508                  db      USB_START_FLAG,11,0
008004 0004          02509                  db      usbCmd_DefVirtDrv
008006 0001          02510                  db      1               ; drive 1
008008 0004          02511                  db      4               ; model IV
00800A 000C          02512                  db      ds_MechRdy+ds_ElecRdy+ds_WriteEnb ; Virt Drv State Flags
00800C 0000          02513                  db      0               ; Virt Drv State LRC
00800E 0000 0000     02514                  db      0,0,0,0         ; Virt Drv State Tape Position (int)
008012 0000          02515                  db      0               ; CRC
008014 00F1          02516                  db      USB_STOP_FLAG
                     02517 ;       Set up a diagnostic loopback
008016 0DF0 0000     02518                  db      USB_START_FLAG,13,0
00801A 0010          02519                  db      usbCmd_DiagUpldLp
00801C 0001          02520                  db      1               ; drive address
00801E 0001          02521                  db      1               ; initial delay in seconds
008020 0001          02522                  db      1               ; repeat delay in hundreds of milliseconds
008022 0000          02523                  db      0,0             ; repeat count, 0 means do this only once (short)
008024 0201 0403 0005 02524                 db      1,2,3,4,5       ; record_data
00802A 0000          02525                  db      0               ; CRC
00802C 00F1          02526                  db      USB_STOP_FLAG
                     02527 ;       Set up a upload record
00802E 04F0 0000     02528                  db      USB_START_FLAG,4,0
008032 0008          02529                  db      usbCmd_UpldRecord
008034 0001          02530                  db      1               ; drive address
008036 0000          02531                  db      0               ; CRC
008038 00F1          02532 UsbFakeE         db      USB_STOP_FLAG
                     02533                  END
```