# ACCESS.bus mouse application code for the microcontroller

# AN445

## DESCRIPTION

ACCESS.bus is an open standard, defining a system for connecting a number of relatively low speed peripheral devices to a host computer, typically a desktop system. The ACCESS.bus (A.b) standard is driven by the increasing demand of workstation and PC users for more peripherals on the desktop than ever before. Devices range from keyboards, mice and trackballs to hand held scanners, card readers and 'virtual reality' gloves. Some of the problems the A.b standard addresses are: difficulty of linking peripherals by non-expert users, desktop wiring clutter, limited number of I/O ports on a workstation, peripheral compatibility with different platforms and the high cost of software driver development associated with adding new peripherals to a system.

At the hardware level, the A.b is based on the $I^2C$ serial bus developed by Philips. The $I^2C$ protocol is supported by standard IC components, including a range of microcontrollers of the 80C51 family. These microcontrollers provide the intelligence for executing the A.b protocol in both peripheral devices and host systems. Many desktop peripherals can be implemented with a single, low cost 8XC751 microcontroller where the firmware supports both the I/O activity and the A.b protocol implementation.

This application note shows the 8XC751 firmware of Digital Equipment Corporation's A.b mouse implementation. Many A.b desktop devices could be implemented with a very similar code. After some discussion of mouse operation we shall give a short overview of the A.b protocol. Our discussion of the A.b is by no means complete—please refer to the specifications for more detailed information.

## MOUSE OPERATION

The mouse is the most popular pointing device for interactive operation with a workstation, personal computer or Windows terminal. It reports to the host two dimensional planar movement, and user's activation of two or three buttons.

Many of the mice available today are opto-mechanical, using shaft encoders. As the mouse is moved over its pad, a lightweight rubber ball turns two perpendicular shafts. When the mouse is held with its cable at the top (away from the user), a left-right movement will rotate the 'X' shaft and an up-down movement will rotate the 'Y' shaft. Any diagonal movement will affect both. The shafts rotate slotted encoder disks which intercept light emitted by an LED. For each shaft there are two phototransistors detecting the light, producing two signals which are out of phase by 90 degrees. Figure 1 shows the waveforms produced for one of the shafts when it rotates. The changes in these quadrature signals can be detected to determine the direction of the mouse movement, and its magnitude. The "positive movement" waveforms relate, for example, to a left to right movement in the X direction. Denoting channel samples as 'AB', a transition from a '00' state to '10' shows a positive movement, while a transition from '00' to '01' shows a negative movement.

The resolution of a mouse is determined by the number of changes to the quadrature waveforms produced in a unit length of planar movement. This is determined by the mechanics of the mouse, regardless of the speed in which the mouse is being moved. The mouse is an incremental pointing device, giving the host periodical position reports which show the displacement change relative to the last report. The microcontroller in the mouse takes the burden of keeping track of the rapid quadrature waveform changes and computing the relative displacement accumulated for each new position report. The quadrature waveforms are sampled, the changes are determined to be positive or negative, and X and Y relative displacement accumulators are being incremented or decremented accordingly.

The average rate of change is determined by the speed of mouse movement. For accurate position reports the encoder waveforms should be sampled frequently enough in order not to miss changes. The DEC mouse produces 200 changes for one inch of movement. Mouse movement at 10 inches per second will yield event rate of 2000 per second, and the microcontroller attempts to sample the encoder waveforms with at least twice that rate—no more than 250 µS between samples. The MAIN routine of the example program performs this sampling in an infinite loop. It reads the position detectors at port 3, compares it to prior readings and if there was a change computes the new value of the relative displacement accumulators YCOUNT and XCOUNT .

Position reports are sent to the host at a much slower rate. In this example, Timer0 interrupts the code at the reporting intervals, and its interrupt routine ("Timer0") initiates a message transmission to the host with the latest information if there was some change in the mouse position or the buttons. The Timer0 service routine samples the position of the three mouse buttons sensed on port 1. Button changes are reported to the host in the same message as the position reports.

## ACCESS.BUS PROTOCOL OVERVIEW

The A.b communications protocol is layered in three levels. The lowest level is a subset of the Philips Inter-integrated Circuit ($I^2C$) bus protocol, above it the A.b Base Protocol common to all types of A.b devices, and on top are the Application Protocols which define message semantics that are specific to particular functional types of devices.

The $I^2C$ protocol defines the low level transaction over the $I^2C$ serial bus, using a single data line (SDA) and a clock line (SCL). The hardware definition for the A.b includes a four wire cable comprised of SDA, SCL and two voltage supply lines. The $I^2C$ provides for cooperative synchronization of the serial clock, bus arbitration, addressing, byte framing and byte acknowledgement by the receiver. The $I^2C$ is a multimaster protocol, and in ACCESS.bus subset the transmitter is always a master. The $I^2C$ allows 128 7 bit addresses, of which 125 may be used in A.b for peripheral microcontrollers. The $I^2C$ protocol burden is typically handled by microcontrollers both at the peripherals and at the host.

The Base Protocol establishes the A.b characteristics including message envelope format, predefined control and status messages, configuration process and the special role of the host. The host acts as a manager of the bus, and all data communication is between the host and peripheral devices—there are no message transactions between peripherals. In A.b, masters are exclusively senders and slaves are exclusively receivers. The host and the attached devices assume master or slave roles at the proper time.

An A.b message is an $I^2C$ bus transaction—a string of bytes sent by a master transmitter where each byte is acknowledged by the slave receiver, and the whole transaction is delimited by Start and Stop conditions. The minimum length of a message is four bytes, and the format definition includes specific locations for source address, destination address, message length and checksum. A protocol flag bit specifies whether the message is a device data stream or a control/status message.

The configuration process is designed to permit auto addressing and hot-plugging of devices. This process detects what devices are present on the bus, assigns unique bus addresses to the attached devices and connects them with the appropriate bus drivers. The configuration process is supported by eight pre-defined control/status

messages. In any A.b system the host address is always the same (50H). When the system is powered up all the peripherals perform self testing, assume a default address (6EH) and send to the host an Attention message announcing their presence. The host sends to each device an identification request message, to which the devices respond with a unique 28 byte ID string. Having received the ID string, the host assigns to each device a unique address. In the case of hot-plugging, the peripheral device and the host will interact in a manner similar to the message exchange during system power up.

In the last phase of the configuration process the host interrogates each device for its "capabilities string"—which describes the functional characteristics and the potential operating modes of the A.b peripheral. Capabilities information allows the software to recognize and use bus devices without additional knowledge about their specific implementation. Using the capabilities information enables writing 'generic' software drivers that can support a range of similar devices, providing some level of device independence and modularity. The capabilities information is transferred to the host as a readable ASCII string with a simple syntax.

A.b application protocols are specific to particular types of devices. The initial A.b specifications define Application protocols for three classes of peripherals: keyboards, locators and text devices. Each class is relatively broadly defined, leaving room for a variety of different devices. When drivers in the operating software of the host fully support a certain class, all devices conforming to the relevant Application Protocol will be supported, without any need for a special software driver.

The Application protocols already defined can support many standard desktop peripherals. The Keyboards protocol supports up to 255 keys. Locator devices can have up to 15 degrees of freedom and up to 16 binary keys or buttons. This can cover devices like mouse, tablet, trackball, 'virtual reality' pointing gloves, dial boxes and function key boxes. The Text Device protocol supports devices that transmit or receive messages consisting of strings of characters in some fixed character set. The simple protocol allows high level flow control, and is appropriate for devices like barcode readers, printers and magnetic card readers.

Each of the Application Protocols has its own set of control/status messages in addition to the predefined messages of the Base Protocol.

## I²C Protocol Handling

The I²C hardware interface on the 8XC751 operates on a bit by bit basis, and the full I²C protocol is supported by a combination of hardware and firmware. This arrangement results in a very compact hardware circuitry necessary for a low cost integrated circuit. The hardware activates and monitors the SDA and SCL lines, performs the necessary arbitration and framing errors checks, and takes care of clock stretching and synchronization. The hardware is synchronized to the software either through polled loops or interrupts. An I²C interrupt is usually requested (if enabled) when a rising edge of SCL indicates a new data on the bus (DRDY), or when a special condition occurs: a frame Start (STR), Stop (STP) or an arbitration loss (ARL). The interrupt is caused by the ATN flag, which is turned on by any of the interrupt inducing conditions. The ATN flag can be polled in a software loop as well.

The example code handles the I²C protocol from an interrupt service routine (ISR). Typically, processing of a frame will be started with an interrupt (at the I2CINT label). If the bus operates at full speed, firmware processing inside a frame will be synchronized to the hardware bit by bit by a polling loop. The firmware polls the ATN flag in a loop limited to about 50 us (WaitATN) whenever it expects something to happen on the bus. If nothing happens during this period of time, the ISR is exited with the I²C interrupt re-enabled. When some bus event will occur later on, processing will resume with a new interrupt.

Processing of bus events monitored by the polling loop is identical to processing events detected by an interrupt. The context from which the mouse was sending or receiving a message is maintained between events (ATN flag activations), and is not lost when exiting the interrupt service routine. The I2Ccxt byte stores the event that is expected, like waiting to send a bit or waiting for an acknowledge. Other I²C context elements are the data byte currently in the send or receive process (I2CDat), a bit counter (BitCnt) keeping track of the location within that data byte and a message byte counter (ByteCnt).

In addition to the parameters that maintain the context of the very 'generic' I²C communications, the code maintains some additional context elements that are relevant to the higher level A.b protocol. These are the computed checksum (Check), the type of message or command being received (RcvType), the type of message being sent or pending (SndType) and a flag indicating that a Position Report transmission is pending (SendRpt).

The Interrupt service routine proceeds in handling the low level details of the I²C protocol as a Slave receiver or a Master transmitter. The routines for Slave or Master processing are separate, and the jump to either one from DISPAT in I2CDONE routine is determined by the MST bit of the I2CON hardware register. The code examines the flags determining which event caused the ATN and then handles the low level hardware according to the context, performing actions like reading a new bit, acknowledging, sending a bit, issuing a Stop and so forth.

When the low level slave receiver code completes reception of a byte, it calls the DORXB routine which deals with the contents of the byte—"application level" routine. Upon return from DORXB there is a call to the Sample subroutines. We effectively sample the quadrature waveforms in between I²C words in order to comply with the requirement for minimum sampling interval. It is interesting to note that code design does not completely separate application code from the I²C low level code—we call Sample from an I²C reception routine (and we do the same in the Master transmission routine). This is because in the 8XC751 the I²C bit processing cannot be done in parallel to other firmware activities and we have to make sure that the application's timing requirements are not being violated.

The Master code will start sending a message if the processing routine was entered due to a Start condition. The routine, in fact, fulfills a request that was issued somewhere else in the code. For example, Timer 0 ISR sets the MASTRQ bit of the I2CON register, and sets the SendRpt flag. MASTRQ causes the processor to seize the bus when it is free and issue a Start. The Master processing routine examines the SendRpt flag, and if it is set the routine will start sending a Position Report.

In a structure similar to the Slave code bit level details are handled in the MASTER routine. Byte transmissions are set up in the DOTXB routine.

## Processing At The ACCESS.bus Protocol Level

A control/status message from the host is identified by the Protocol Flag, the most significant bit of the third message byte. The message body is a code for the command. When such messages are received, they are processed by the DORXCMD routine. Control/Status messages can be of either the Base Protocol or the Application Protocol. In the listing, base protocol codes have the prefix 'I_', and application level protocol

commands has the prefix ‘App_’ (the definitions are in the include file ‘ab.inc’).

The Base Protocol commands from the host are I_Reset, I_IdReq, I_ASgnAdr and I_CapReq. During the configuration process the mouse responds to the host with device to host control/status messages: I_Attn, I_IdReply, I_CApReply and I_Error.

The string for the I_IdReply message is defined in GET_ID. The module revision and vendor name are padded with space characters in order to fit the fixed string length. The last four bytes of the ID string are a device number that can distinguish otherwise like devices with the same firmware. The protocol allows it to be a serial number or a pseudo random number. Our mouse uses a pseudo random number, produced by reading the 16 bit contents of Timer0 that is active since power-up (the number is extended to 32 bits by appending FFFF). The protocol includes ‘protection’ against the rare event in which two like devices report the same pseudo random number or are mistakenly assigned to the same address. Just prior to sending the first data message to the host, each interactive device transmits a Reset message to its own assigned address (see PosMsg label in the example code). Any other device with the same address will be forced to the power-up default address and will undergo configuration again, as it was hot-plugged onto the bus.

The Capabilities String for the I_CapReply message is defined in GET_CAP. The string identifies the device as a mouse with specific characteristics: three buttons, two dimensions, relative location reports with 200 dpi resolution etc. The ‘prot(locator)’ element tells the A.b software driver to use the Locator Device Protocol.

The Locator Device Protocol is one of three application protocols already defined for the highest layer of the A.b protocol. This protocol defines a “Locator Event Report” which is used for the Position Reports of the mouse.

A Locator Event Report is sent in the format of the device data stream Message defined in the base A.b protocol. The message body includes the current state of the buttons and the location difference from the last report. This data is coded as a sequence of two byte integers. For the mouse which is a two dimensional device, the message data stream length is six bytes, or three integers. The first integer contains the state of 0-15 locator key switches. For the three button mouse, only three of these sixteen bits carry meaningful information. The remaining integers represent the position of the locator dimensions—the contents of the X and Y displacement accumulators.

Three control messages specific to the Locator Protocol are processed at DORXCMD. The host initiates a self test by App_Test. App_Poll initiates one time transmission of a position report, and App_Setinterval modifies the default reporting interval controlled by the reload value of Timer0.

This note highlights some of the implementation details—the commented listing covers the rest. As one can see, the A.b protocols are relatively simple to program in firmware. The low-level $I^2C$ implementation on the 8XC751 is somewhat involved, but the same low level routines can be re-used for different devices.

The source code files for this program are available for download from the Philips Semiconductors computer bulletin board system. This system is open to all callers, operates 24 hours a day, and can be accessed with modems at 2400, 1200, and 300 baud. The telephone numbers for the BBS are: (800) 451-6644 (in the U.S. only) or (408) 991-2406.

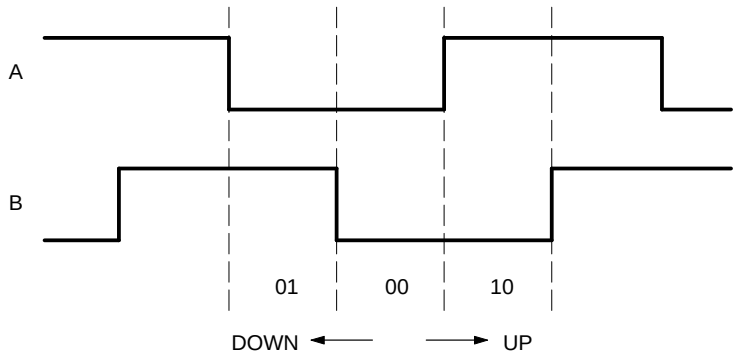


**Figure 1. Example of Quadrature Encoding Waveforms**

```
MS-DOS MACRO ASSEMBLER A51 V4.4
OBJECT MODULE PLACED IN MOUSE.OBJ
ASSEMBLER INVOKED BY:  A51 MOUSE.A51
LOC  OBJ             LINE     SOURCE
                        1     ;************************************************
                        2     ; Module: mouse.a51
                        3     ;
                        4     ; Firmware design and code for I2C desktop bus Mouse
                        5     ;   Environment:  83C751  Assembler
                        6     ;   Author:       Robert Clemens   10-Jul-1990
                        7     ;                 (I2C I/O adapted from P.Sichel's "Keyboard" code)
                        8     ;   Revision:     6-Mar-1991
                        9     ;
                       10     ;   31-Jan-1991 PAS  Add numerous keyboard fixes.
                       11     ;                    Streamline input sample and I2C code.
                       12     ;                    Separate HW dependent constants.
                       13     ;                    Fix RxEnable after bus time out.
                       14     ;
                       15     ;   06-Feb-1991 PAS  Rev X0.6
                       16     ;                    Fix sample timer initialization, use 14ms as default.
                       17     ;                    Fix length checking to allow commands with
                       18     ;                      more parameters than required.
                       19     ;                    Implement Set Interval command.
                       20     ;                    Handle LLLLL=0 to mean 32.
                       21     ;                    Fix ARL during self-addressed reset message.
                       22     ;                    Fix to handle DRDY and ARL together.
                       23     ;                    Re-order mouse buttons as MRL, update Capabilities.
                       24     ;                    Do not allow other interrupts during TimerI svc.
                       25     ;                    Document sampling requirements.
                       26     ;                    Document hardware details.
                       27     ;                    Add check to skip waiting after DNRXB.
                       28     ;                    Misc clean-up in: BeMast, Assign,...
                       29     ;                    Use include files for 751 registers and ODB msgs.
                       30     ;
                       31     ;    6-Mar-1991 PAS  Rev X0.7
                       32     ;                    Fix MN8Bit to check ARL before clearing DRDY.
                       33     ;                    Separate SendRpt flag from movement detected (Movement)
                       34     ;                       so only Timer0 will initiate motion reports.
                       35     ;                    Don't send Position report to def_addr even if polled.
                       36     ;                    Report InputError for invalid checksum,
                       37     ;                       unrecognized command code, or illegal parameter
                       38     ;                       value.  Do not complain about parameters beyond
                       39     ;                       those anticipated.
                       40     ;                    Sample quadrature inputs between I2C bytes
                       41     ;                       to insure accurate tracking.
                       42     ;
                       43     ;   13-Mar-1991 PAS  Fix DoStp4 to borrow Rx code and become IDLE rcv.
                       44     ;
                       45     ;   26-Mar-1991 PAS  Rex X0.8
                       46     ;                    Update to use 4-byte device number.
                       47     ;                    Get new device number only after Reset.
                       48     ;
                       49     ;    8-Apr-1991 PAS  Add error checking to CapReq.
                       50     ;
                       51     ;    9-Jul-1991 PAS  Add protocol_revision as part of module revision.
                       52     ;
                       53     ;   29-Jul-1991 PAS  Protocol_revision in 1-byte, fix ARL and MASTER bug.
                       54     ;                    Ignore unrecognized commands.  Add I_MsgCheck.
                       55     ;
                       56     ;   11-Sep-1991 PAS  Rev X1.2.  Identify button positions as 1,2,3.
```

```
LOC  OBJ            LINE     SOURCE
                       57     ;                    Use new ab.inc file.  Change I_MsgCheck to
                       58     ;                    I_Error and do not overwrite pending SndType.
                       59     ;                    Retry message after Negative Ack (NACnt).
                       60     ;
                       61     ;    7–Oct–1991 PAS  Rev X1.3.  Fix spurious large count problem.
                       62     ;                    Fix STOP detected while MASTER without ARL.
                       63     ;
                       64     ;   25–Oct–1991 PAS  Rev X1.4.  Improve TimerI handler to avoid
                       65     ;                    lockup when MASTRQ with SCL low.
                       66     ;
                       67     ;    4–Nov–1991 PAS  V1.0 release for Boston mfg.
                       68     ;   22–Dec–1991 PAS  V1.1 align data with tx bit that lost arbitration
                       69     ;
                       70     ;
                       74     ;
                       75     ;***********************************************
                       76     $ TITLE (Digital ACCESS.bus Mouse, V1.1)
                       77     $ DATE (12/22/91)
                       78     $ DEBUG
                       79     $ NOMOD51       ;83C751 is not model 51
                       80                     ;Define SFRs explicitly
                       81
                       82
                       83     ; Symbolic addresses and masks
                       84
                       85     $ INCLUDE( /dskbus/include/arch/reg751.inc )
                 =1    86     ;***********************************************
                 =1    87     ; Module: /dskbus/include/arch/reg751.inc
                 =1    88     ;
                 =1    89     ; 83c751 SFR declarations
                 =1    90     ;   Environment:  83C751 Assembler
                 =1    91     ;
                 =1    95     ;   Date        Revision        Perpetrator
                 =1    96     ;
                 =1    97     ;   30–Jan–91   X0.1            Mark Shepard
                 =1    98     ;       Created (from previous keyboard module)
                 =1    99     ;
                 =1   100     ;
                 =1   101                      =1   102     $EJ
                 =1   103     ;***************
                 =1   104
                 =1   105     ; Interrupt Enable Register
  00A8           =1   106     IE      EQU     0A8h
  0000           =1   107     EX0     EQU     0                       ;External interrupt 0
  0001           =1   108     ET0     EQU     1                       ;Timer0 interrupt
  0002           =1   109     EX1     EQU     2                       ;External interrupt 1
  0003           =1   110     ET1     EQU     3                       ;TimerI interrupt
  0004           =1   111     EI2     EQU     4                       ;I2C interrupt
  0007           =1   112     EA      EQU     7                       ;All interrupt enable/disable bit
                 =1   113
                 =1   114     ; I2C Control Register
  0098           =1   115     I2CON   EQU     098h
                 =1   116     ; Input (read) is bit #s for JB etc...
  0007           =1   117     RDAT    EQU     7                       ;receive data
  0006           =1   118     ATN     EQU     6                       ;attention
  0005           =1   119     DRDY    EQU     5                       ;data ready
  0004           =1   120     ARL     EQU     4                       ;arbitration loss
  0003           =1   121     STR     EQU     3                       ;start
  0002           =1   122     STP     EQU     2                       ;stop
```

```
LOC  OBJ            LINE     SOURCE
  0001          =1   123     MST     EQU     1                       ;master
                =1   124     ; Output (write) is binary values for MOV I2CON,#...
  0080          =1   125     CXA     EQU     80h                     ;clear xmit active
  0040          =1   126     IDLE    EQU     40h                     ;set to idle slave
  0020          =1   127     CDR     EQU     20h                     ;clear data ready
  0010          =1   128     CARL    EQU     10h                     ;clear arbitration loss
  0008          =1   129     CSTR    EQU     08h                     ;clear start
  0004          =1   130     CSTP    EQU     04h                     ;clear stop
  0002          =1   131     XSTR    EQU     02h                     ;transmit start
  0001          =1   132     XSTP    EQU     01h                     ;transmit stop
                =1   133
                =1   134     ; I2C Data Register
  0099          =1   135     I2DAT   EQU     099h
  0080          =1   136     XDAT    EQU     80h                     ;transmit data
                =1   137
                =1   138     ; I2C Configuration Register
  00D8          =1   139     I2CFG   EQU     0D8h
  0080          =1   140     SLAVEN  EQU     80h                     ;enable slave mode
  0007          =1   141     SLAVENB EQU     7
  0040          =1   142     MASTRQ  EQU     40h                     ;master request
  0006          =1   143     MASTRQB EQU     6
  0020          =1   144     CLRTI   EQU     20h                     ;clear timerI
  0005          =1   145     CLRTIB  EQU     5
  0010          =1   146     TIRUN   EQU     10h                     ;timerI run
  0004          =1   147     TIRUNB  EQU     4
                =1   148
  0088          =1   149     TCON    EQU     088h                    ;Timer Control
                =1   150
                =1   151     ; 83C751 SFRs
  0080          =1   152     P0      EQU     080h
  0080          =1   153     SCL     BIT     P0.0
  0081          =1   154     SDA     BIT     P0.1
  0081          =1   155     SP      EQU     081h
  0082          =1   156     DPL     EQU     082h
  0083          =1   157     DPH     EQU     083h
  008A          =1   158     TL      EQU     08Ah                    ;Timer Lo
  008B          =1   159     RTL     EQU     08Bh                    ;Reload TL
  008C          =1   160     TH      EQU     08Ch                    ;Timer Hi
  008D          =1   161     RTH     EQU     08Dh                    ;Reload TH
  0090          =1   162     P1      EQU     090h
  00B0          =1   163     P3      EQU     0B0h
  00D0          =1   164     PSW     EQU     0D0h
  00E0          =1   165     ACC     EQU     0E0h
  00F0          =1   166     B       EQU     0F0h
                =1   167
                =1   168     ;*************** End of include file
                     169
                     170     $ INCLUDE( /dskbus/include/ab.inc )
                =1   171     ;******************module ab.inc*****************************
                =1   172     ;
                =1   173     ; Ab Base Protocol definitions
                =1   174     ;   Environment:  83C751 Assembler
                =1   175     ;
                =1   179     ;   Date        Revision        Perpetrator
                =1   180     ;
                =1   181     ;   04-Sep-91   ----            Mark Shepard
                =1   182     ;       Changed I_MsgCheck to I_Error, kept I_MsgCheck for backpatibility.
                =1   183     ;       Added App_Error (0xb4 to correspond to I_Error).
                =1   184     ;       Changed Sig_Attn from 0 to 3 to make KB code easier.
```

```
LOC  OBJ            LINE     SOURCE
                   =1   185     ;
                   =1   186     ;   04-Aug-91   ----            Mark Shepard
                   =1   187     ;       Renamed to generic "ab.inc"
                   =1   188     ;       Added defines for header structure (mainly for documentation purposes).
                   =1   189     ;       Added defines for bus signal codes (RESET, HALT, ATTN, etc.)
                   =1   190     ;
                   =1   191     ;   22-May-91   X0.2            Peter Sichel
                   =1   192     ;       Added Vendor command codes, and I_MsgCheck.
                   =1   193     ;
                   =1   194     ;   30-Jan-91   X0.1            Mark Shepard
                   =1   195     ;       Created from ODB base protocol spec, X0.7.
                   =1   196     ;
                   =1   197
                   =1   198     $EJ
                   =1   199     ;***************
                   =1   200     ;  Desktop bus command codes for all Interface-part and Application-part
                   =1   201     ;  commands in the Base Protocol spec.
                   =1   202     ;
                   =1   203     ;  Naming Convention -
                   =1   204     ;    Interface-Part codes are prefixed with "I_",
                   =1   205     ;    Application-Part with "App_".  Codes specific to a particular
                   =1   206     ;    sub-protocol (e.g. Keyboard Protocol) could be prefixed with
                   =1   207     ;    "Key_", "Kb_", or "Kbp_".
                   =1   208     ;
                   =1   209     ;    Definitions for sub-protocols should go in separate include files.
                   =1   210     ;    (keyp.inc, locp.inc, textp.inc, etc.).
                   =1   211     ;
  0000             =1   212     I_NoMsg         EQU     000h    ; special value, means not a valid message
                   =1   213
  00F0             =1   214     I_Reset         EQU     0f0h    ; reset device
  00F1             =1   215     I_IdReq         EQU     0f1h    ; Identify request
  00F2             =1   216     I_AsgnAdr       EQU     0f2h    ; assign address
  00F3             =1   217     I_CapReq        EQU     0f3h    ; capabilities (fragment) request
                   =1   218
  00E0             =1   219     I_Attn          EQU     0e0h    ; power-on/reset attention
  00E1             =1   220     I_IdReply       EQU     0e1h    ; identify reply
  00E3             =1   221     I_CapReply      EQU     0e3h    ; capabilities (fragment) reply
  00E4             =1   222     I_Error         EQU     0e4h    ; (the future) interface/bus std error report
  00E4             =1   223     I_MsgCheck      EQU     0e4h    ; *** obsolete, don't use in new code ***
                   =1   224
  00C0             =1   225     I_Vendor0       EQU     0c0h    ; vendor reserved command
  00C1             =1   226     I_Vendor1       EQU     0c1h    ; vendor reserved command
  00C2             =1   227     I_Vendor2       EQU     0c2h    ; vendor reserved command
  00C3             =1   228     I_Vendor3       EQU     0c3h    ; vendor reserved command
                   =1   229
  00A0             =1   230     App_Sig         EQU     0a0h    ; hardware signal
  00A1             =1   231     App_TestReply   EQU     0a1h    ; test reply (for a specific application-part)
                   =1   232
  00B1             =1   233     App_Test        EQU     0b1h    ; self-test result request
  00B4             =1   234     App_Error       EQU     0b4h    ; (the future) std application error report
                   =1   235
                   =1   236     ;***************
                   =1   237     ;  Well-known I2C addresses used by desktop bus peers
                   =1   238     ;
                   =1   239     ; I2C Address Allocation
                   =1   240     ; row        column
                   =1   241     ; A6,A5,A4 / A3,A2,A1,A0            R/W=0 (write)
                   =1   242     ; Host: 2/8                         50h
                   =1   243     ; Devices: 2/9-2/15, 3/0-3/7        52-6Eh (even numbers only)
```

```
LOC  OBJ            LINE     SOURCE
                =1   244     ; Device default: 3/7               6Eh
                =1   245
  0050          =1   246     Adr_Host        EQU     050h    ; standard Host address
  006E          =1   247     Adr_Default     EQU     06eh    ; default (pwr-up) address for peripherals
                =1   248
                =1   249     ;**************
                =1   250     ;  Bus-Signal codes defined at the Base Protocol level
                =1   251     ;
  0001          =1   252     Sig_Reset       EQU     1       ; hardware reset!
  0002          =1   253     Sig_Halt        EQU     2       ; debugging interrupt
  0003          =1   254     Sig_Attn        EQU     3       ; general-purpose interrupt from User
                =1   255
                =1   256     ;**************
                =1   257     ;  Defines related to message structure
                =1   258     ;
                =1   259     ;  Declarations prefixed with "AbWire_" refer to Ab objects, fields, etc.
                =1   260     ;  as transmitted across the i2c "wire" (as opposed to the "optimized frame
                =1   261     ;  format used between the host and 83c751 host-ctrlr).
                =1   262     ;
                =1   263     ;
  0003          =1   264     AbWire_HdrSiz   EQU     3       ; Ab Header Size on the wire:  dst + src + PLen
  007F          =1   265     AbWire_LenMask  EQU     7fh     ; Ab mask for length field
  007F          =1   266     AbWire_MaxLen   EQU     127     ; Ab maximum data bytes in message
                =1   267
                =1   268     ;*************** End of include file
                     269
                     270
                     271     $EJ
                     272     ;*************************************
                     273     ;* Hardware/timing-dependent constants *
                     274     ;*************************************
                     275
                     276     ;CT             EQU     02h     ;CT1, CT0 fmax = 16.8 MHz
                     277     ;CT             EQU     01h     ;CT1, CT0 fmax = 14.25 MHz
                     278     ;CT             EQU     00h     ;CT1, CT0 fmax = 11.7 MHz
  0003               279     CT              EQU     03h     ;CT1, CT0 fmax = 9.14 MHz
                     280
  009A               281     IntEnab         EQU     09Ah    ;enable EA+EI2+ET1+ET0.
  0010               282     INIT_TCON       EQU     010h    ;Timer0 init for internal operation.
                     283
  00DB               284     DEF_RTH         EQU     0DBh    ;Sampling interval (14ms with 8 MHz clock).
  008B               285     DEF_RTL         EQU     08Bh    ; Used as default Timer0 reload value.
  0002               286     MSECH           EQU     002h    ;Timer offset for 1 ms with 8 MHz clock.
  009A               287     MSECL           EQU     09Ah    ; 029Ah=666 * 3/2 mms/tick = 1000 ticks = 1ms.
                     288
  0008               289     DelayATN        EQU     8       ;about 50µS (wait for I2C activity)
                     290
  0010               291     CapFragLen      EQU     16      ;Capabilities fragment length.
                     292
                     293     ; *** Hardware Interface Notes ***
                     294     ; P3 is used to read the X-Y quadrature inputs.
                     295     ;   P3.0 = XB
                     296     ;   P3.1 = XA
                     297     ;   P3.2 = YA
                     298     ;   P3.3 = YB
                     299     ;
                     300     ; Notice P3.0 is connected to the B side of the X encoder
                     301     ;  while P3.2 is connected to the A side of the Y encoder.
                     302     ; This is to compensate for the orientation of the inclined cylinders.
```

```
LOC  OBJ            LINE     SOURCE
                     303     ; Positive X movement causes clockwise rotation of the encoder shaft.
                     304     ; Positive Y movement causes counter clockwise rotation of the encoder shaft.
                     305     ;
                     306     ; P1 is used to read the switch inputs.
                     307     ;   P1.0 = middle mouse button.
                     308     ;   P1.1 = left mouse button.
                     309     ;   P1.2 = right mouse button.
                     310     ;
                     311     ; 6–Feb–1991  This order of buttons reflects the current PCB layout
                     312     ;             and is subject to change.
                     313
                     314     $EJ
                     315     ; Locator messages (device defined)
                     316     ;   send types
  0003               317     LD_Position     EQU     3       ;Device state report
                     318
                     319     ;   receive types
  00B0               320     LD_Poll         EQU     0B0h
  0082               321     LD_SetInterval  EQU     082h
                     322
                     323
                     324     ; SelfTest errors (0=success)
  0000               325     NO_ERROR        EQU     0
  0001               326     ROM_ERROR       EQU     1
  0002               327     RAM_ERROR       EQU     2
                     328
                     329
                     330     $EJ
                     331     ;************
                     332     ; RAM usage *
                     333     ;************
                     334     ; I2C variables
                     335     ; Register bank 0
                     336                                 ;R0 – command parameter
                     337                                 ;R1 – index pointer
  0002               338     I2CDat   DATA   02h         ;The byte being sent or received.
  0003               339     BitCnt   DATA   03h         ;I2C bit counter
  0004               340     ByteCnt  DATA   04h         ;I2C message byte counter
  0005               341     ATNCnt   DATA   05h         ;ATN Retry counter
  0006               342     I2CCxt   DATA   06h         ;I2C context, the event the CPU is waiting for.
  0007               343     Temp     DATA   07h         ;All purpose temp
                     344
                     345     ; Desktop Bus Protocol
  0008               346     MyAddr   DATA   08h         ;I2C address assigned this device.
  0009               347     NACnt    DATA   09h         ;Negative Ack retry counter.
  000A               348     RcvType  DATA   0Ah         ;Message or command type being received.
  000B               349     SndType  DATA   0Bh         ;Message type being sent, or pending.
  000C               350     MsgLen   DATA   0Ch         ;Message length field.
  000D               351     Check    DATA   0Dh         ;Message checksum.
  000E               352     RandH    DATA   0Eh         ;Random number (2 bytes)
  000F               353     RandL    DATA   0Fh
                     354
                     355     ; Locator report buffer
  0010               356     ReportBuf       EQU     10h     ;Beginning of position report buffer.
  0010               357     Switch2         DATA    10h     ;Switch data (Buttons 9 to 16)
  0011               358     Switch1         DATA    11h     ;Switch data (Buttons 1 to 8 )
  0012               359     XBUF2           DATA    12h     ;XData transmission buffer (MSB)
  0013               360     XBUF1           DATA    13h     ;XData transmission buffer (LSB)
  0014               361     YBUF2           DATA    14h     ;YData transmission buffer (MSB)
```

```
LOC  OBJ            LINE     SOURCE
  0015                362     YBUF1           DATA    15h     ;YData transmission buffer (LSB)
                      363
  0016                364     XCOUNT          DATA    16h     ;XData
  0017                365     YCOUNT          DATA    17h     ;YData
  0018                366     MARKER          EQU     YCOUNT+1
                      367
  0018                368     CapOffset       DATA    18h     ;Capabilities fragment offset
  0019                369     CapLen          DATA    19h     ;Capabilities fragment length
  001A                370     SelfTest        DATA    1Ah     ;Self Test result variable location
  001B                371     RomSum          DATA    1Bh     ;Hold ROM checksum
  001C                372     SampleClock     DATA    1Ch     ;Time stamp of last sample
                      373                                     ;3 spare.
                      374
                      375     ; Bit addressable area begins at 20h
                      376     ; Location and switch compare variables
  0020                377     LastXY          DATA    20h     ;Last X/Y Position  (from P3)
  0021                378     LastSW          DATA    21h     ;Last Switch Status (from P1)
  0022                379     TranXY          DATA    22h     ;Port 3 transition register (bit addressable)
                      380
                      381     $EJ
                      382     ; I2C status and position scanning flags.
  0023                383     Flags           DATA    23h
  0018                384     Prot            BIT   Flags.0   ;1=C/S message; 0=device data stream.
  0019                385     SendRpt         BIT   Flags.1   ;New Position Report flag.
  001A                386     Movement        BIT   Flags.2   ;Movement detected flag.
  001B                387     RxEnable        BIT   Flags.3   ;I2C receive enable.
  001C                388     TxSelfRst       BIT   Flags.4   ;Indicates send Self–Reset after Assign (0).
  001D                389     KeepID          BIT   Flags.5   ;Set means keep same device number.
  001E                390     NotMyID         BIT   Flags.6
  001F                391     AA              BIT   Flags.7   ;Assert Acknowledge.
                      392
  0024                393     FlagsA          DATA    24h
  0020                394     MsgCheck        BIT   FlagsA.0  ;I2C message checksum or framing error.
                      395
                      396                                     ;11 spare
                      397
  0030                398     StackBase       DATA    30h     ;16 byte stack
                      399                                     ; need 2 bytes per subroutine
                      400                                     ; 4 bytes per interrupt (max 2)
                      401     ;RAM ends at 3Fh
                      402     ; 50 bytes RAM used, 64 maximum.
                      403
                      404
                      405     $EJ
                      406     ;******************
                      407     ; Code begins
                      408     ;******************
0000                  409             ORG     0       ;Power up reset vector.
0000 01B4             410             AJMP    PwrUp
                      411
0003                  412             ORG     003h    ;INT0 is not used.
0003 01B4             413             AJMP    PwrUp
                      414     ;
                      415     ; Use this spare byte to hold the ROM checksum complement.
0005 51               416             DB      051h
                      417
000B                  418             ORG     00Bh    ;Timer0 interrupt vector
000B 0125             419             AJMP    Timer0  ;16 bit system tick generator
                      420
```

```
LOC  OBJ            LINE     SOURCE
0013                 421             ORG     013h    ;INT1 is not used.
0013 01B4            422             AJMP    PwrUp
                     423
001B                 424             ORG     01Bh    ;TimerI interrupt vector
001B 014F            425             AJMP    TimerI  ;I2C time out timer
                     426
0023                 427             ORG     023h    ;I2C interrupt vector
0023 21D8            428             AJMP    I2CINT
                     429
                     430
                     431     $EJ
                     432     ;*****************************
                     433     ; Timer0 Interrupt
                     434     ; Position sampling interval time out.
                     435     ;*****************************
                     436
0025 C0D0            437     Timer0: PUSH    PSW                     ;Save registers we need.
0027 C0E0            438             PUSH    ACC
0029 E508            439             MOV     A,MyAddr                ;Check for default address.
002B B46E02          440             CJNE    A,#Adr_Default,AddrOK
002E 801A            441             SJMP    T0Exit                  ;Don't send Position reports
                     442                                             ; to default address.
0030 E590            443     AddrOK: MOV     A,P1                    ;Get switch info from P1
                     444                                             ; (0=button depressed).
0032 F4              445             CPL     A                       ;Complement
0033 5407            446             ANL     A,#00000111b            ;We need low 3 bits of P1
0035 B52105          447             CJNE    A,LastSW,CHNSWI         ;if new, switches did change
                     448             ;Switches did not change, check movement.
0038 301A0F          449             JNB     Movement,T0Exit
003B 8009            450             SJMP    T0Send                  ;Yes, go send report.
                     451
003D F521            452     CHNSWI: MOV     LastSW,A                ;Save LastSW for next compare.
                     453             ;Re-order switches from RLM to MRL until PCB is fixed.
003F 13              454             RRC     A
0040 92E2            455             MOV     ACC.2,C
0042 5407            456             ANL     A,#00000111b
0044 F511            457             MOV     Switch1,A               ;Move to output buffer.
                     458
0046 D219            459     T0Send: SETB    SendRpt                 ;Set to send Position report.
0048 D2DE            460             SETB    I2CFG.MASTRQB           ;Request to be master.
                     461
004A D0E0            462     T0Exit: POP     ACC
004C D0D0            463             POP     PSW
004E 32              464             RETI
                     465
                     466     $EJ
                     467     ;*******************************
                     468     ; TimerI interrupt
                     469     ; The I2C bus has timed out,
                     470     ; no SCL for at least 1020 machine cycles during an active frame.
                     471     ; Since SCL is stuck, we can't wait for DRDY.
                     472     ; Try to fix it manually.
                     473     ;*******************************
                     474
004F C2AF            475     TimerI:   CLR   IE.EA           ;Disable interrupts.
0051 75D823          476               MOV   I2CFG,#CLRTI+CT ;Clear interrupt and turn off TimerI.
                     477                                     ; manually clear SLAVEN & MASTRQ.
0054 7598BC          478               MOV   I2CON,#CXA+CARL+CDR+CSTR+CSTP   ;Clear I2C flags.
0057 758130          479               MOV   SP,#StackBase   ;Reset SP.
```

```
LOC  OBJ            LINE     SOURCE
                     480
                     481     ; Attempt to regain control of the I2C bus after a bus fault.
005A D280            482     FixBus:   SETB  SCL             ;Insure I/O port is not locking I2C.
005C D281            483               SETB  SDA
005E 308020          484               JNB   SCL,ResetBus    ;If SCL is low, bus cannot be fixed.
0061 208113          485               JB    SDA,RStop       ;If SCL & SDA are high, force a stop.
                     486               ;SDA is low, attempt to release SDA by clocking SCL.
0064 750309          487               MOV   BitCnt,#9       ;Set max # of tries to clear bus.
0067 C280            488     ClockBus: CLR   SCL             ;Force an I2C clock.
0069 11AF            489               ACALL SDelay
006B 208109          490               JB    SDA,RStop       ;Did it work?
006E D280            491               SETB  SCL
0070 11AF            492               ACALL SDelay
0072 D503F2          493               DJNZ  BitCnt,ClockBus ;Repeat clocks until SDA clears or retry limit.
0075 800A            494               SJMP  ResetBus        ;Failed to fix bus by this method.
                     495
0077 C281            496     RStop:    CLR   SDA             ;Try forcing a stop since
0079 11AF            497               ACALL SDelay          ; SCL & SDA are both high.
007B D280            498               SETB  SCL
007D 11AF            499               ACALL SDelay
007F D281            500               SETB  SDA
                     501     ;
0081                 502     ResetBus: ;Wait for bus to clear.
0081 3080FD          503               JNB   SCL,$
0084 3081FD          504               JNB   SDA,$
0087 7401            505               MOV   A,#1                    ;Pause for bus to stabilize.
0089 11A7            506               ACALL LDelay
                     507               ;Re-enable I2C functions.
008B 750B00          508               MOV   SndType,#I_NoMsg        ;Cancel message if any.
008E D21B            509               SETB  RxEnable                ;Enable receiving.
0090 1194            510               ACALL InitI2C                 ;Initialize I2C.
0092 2154            511               AJMP  MAIN                    ;Restart MAIN.
                     512
                     513
                     514     ; Initialize I2C functions
0094 75D893          515     InitI2C: MOV    I2CFG,#SLAVEN+TIRUN+CT  ;Enable I2C
                     516             ;Set I2C to be idle receiver & clear all flags.
0097 7598FC          517             MOV     I2CON,#CXA+IDLE+CDR+CARL+CSTR+CSTP
009A 750601          518             MOV     I2CCxt,#RXIDLE          ;Context idle receiver
009D 31F3            519             ACALL   XRETI                   ;Clear pending interrupt if any.
009F 75A89A          520             MOV     IE,#IntEnab             ;Enable interrupts (EA+EI2+ET1+ET0)
00A2 7410            521             MOV     A,#16                   ;Wait to sync message frame.
00A4 11A7            522             ACALL   LDelay
00A6 22              523             RET
                     524
                     525     ; Long Delay, A/2 milliseconds.
00A7 7FA6            526     LDelay: MOV     R7,#166
00A9 DFFE            527             DJNZ    R7,$
00AB D5E0F9          528             DJNZ    ACC,LDelay
00AE 22              529             RET
                     530
                     531     ; Short delay routine (10 machine cycles).
00AF 11B1            532     SDelay:   ACALL SD1
00B1 00              533     SD1:      NOP
00B2 00              534               NOP
00B3 22              535               RET
                     536
                     537     $EJ
```

```
LOC  OBJ            LINE     SOURCE
                      538     ;*********************************
                      539     ; Power up initialization starts here
                      540     ;*********************************
                      541     ; Reset command branches to here.
00B4 75086E           542     PwrUp:  MOV     MyAddr,#Adr_Default     ;Re-initialize default address.
00B7 E4               543             CLR     A
00B8 F5A8             544             MOV     IE,A            ;Disable interrupts.
00BA 75D803           545             MOV     I2CFG,#CT       ;Disable I2C
00BD 53D0E7           546             ANL     PSW,#0E7h       ;Select RB0.
00C0 758130           547             MOV     SP,#StackBase
                      548             ; Initialize I/O pins
00C3 758003           549             MOV     P0,#03h         ;Initialize I2C I/O pins, SCL & SDA high
00C6 7590FF           550             MOV     P1,#0FFh        ;P1 set for input to read switches.
00C9 75B0FF           551             MOV     P3,#0FFh        ;P3 set for input to read X-Y.
                      552             ; Initialize Timer0
00CC 758DDB           553             MOV     RTH,#DEF_RTH    ;14 ms at 8 MHz
00CF 758B8B           554             MOV     RTL,#DEF_RTL
00D2 758810           555             MOV     TCON,#INIT_TCON ;Running, internal mode clock/12.
                      556
                      557     ;************
                      558     ; Perform ROM Test (0-7FFh, 2K bytes)
                      559     ;************
00D5 75F000           560     TestROM:MOV     B,#0            ;Initialize sum
00D8 900000           561             MOV     DPTR,#0000h     ;Set pointer to start of ROM
00DB C3               562             CLR     C
00DC E4               563     SumLp:  CLR     A
00DD 93               564             MOVC    A,@A+DPTR       ;Get byte from ROM
00DE 35F0             565             ADDC    A,B             ;Add sum
00E0 F5F0             566             MOV     B,A             ;Save sum in B
00E2 A3               567             INC     DPTR
00E3 E583             568             MOV     A,DPH           ;Check if ROM complete
00E5 B408F4           569             CJNE    A,#08h,SumLp
00E8 5002             570             JNC     TestSum         ;Add carry if set
00EA 05F0             571             INC     B
00EC E5F0             572     TestSum:MOV     A,B
00EE 6007             573             JZ      TestRAM         ;If zero, ROM is Okay
00F0 751A01           574             MOV     SelfTest,#ROM_ERROR
00F3 F51B             575             MOV     RomSum,A        ;Save bad checksum.
00F5 8023             576             SJMP    BadMem1
                      577
                      578     $EJ
                      579     ;************
                      580     ; Perform RAM test (0-3Fh, 64 bytes)
                      581     ; Does not test special function registers.
                      582     ; A, B, and R0 are not preserved.
                      583     ;************
00F7 78AA             584     TestRAM: MOV    R0,#0AAh        ;Test RAM location 0.
00F9 B8AA1B           585             CJNE    R0,#0AAh,BadMem
00FC 7855             586             MOV     R0,#055h
00FE B85516           587             CJNE    R0,#055h,BadMem
0101 783F             588             MOV     R0,#3Fh         ;Init R0 to top of RAM.
0103 74AA             589             MOV     A,#0AAh         ;Test alternate bits.
0105 86F0             590     ChkRAM: MOV     B,@R0           ;Save previous contents.
0107 F6               591             MOV     @R0,A
0108 B6AA0C           592             CJNE    @R0,#0AAh,BadMem
010B 23               593             RL      A               ;Test other bits.
010C F6               594             MOV     @R0,A
010D B65507           595             CJNE    @R0,#055h,BadMem
0110 23               596             RL      A
```

```
LOC  OBJ            LINE     SOURCE
0111 A6F0            597             MOV     @R0,B           ;Restore contents.
0113 D8F0            598             DJNZ    R0,ChkRAM
0115 8005            599             SJMP    MemOK
0117 751A02          600     BadMem: MOV     SelfTest,#RAM_ERROR
                     601
                     602     ; Report bad memory.  Since a memory problem was detected, the
                     603     ; normal I2C transmit code may be unreliable.  Hope it isn't
                     604     ; a fatal problem and use it anyway.  There's only so much
                     605     ; we can do.  Could add special code here.
011A 8005            606     BadMem1: SJMP   InitRAM
                     607
011C 751A00          608     MemOK:  MOV     SelfTest,#0
011F 8000            609             SJMP    InitRAM
                     610
                     611
                     612     $EJ
                     613     ;************
                     614     ;Initialize RAM
                     615     ;************
0121 E4              616     InitRAM:CLR     A                       ;Init Acc to Zero
0122 F50A            617             MOV     RcvType,A               ;Clr RcvType
0124 F50B            618             MOV     SndType,A               ;Clr SndType
0126 7910            619             MOV     R1,#ReportBuf           ;Clear report buffer & compare vars
0128 F7              620     ClrBuf: MOV     @R1,A                   ;Clear location
0129 09              621             INC     R1                      ;Go to next location
012A B918FB          622             CJNE    R1,#Marker,ClrBuf       ;Check for end
012D F523            623             MOV     Flags,A                 ;Init Flags
012F F521            624             MOV     LastSW,A                ;Init last switch image.
0131 750905          625             MOV     NACnt,#5                ;Negative Ack retry count.
                     626
                     627     ;Init LastXY
0134 E5B0            628             MOV     A,P3                    ;Read X-Y quad inputs to init LastXY.
0136 540F            629             ANL     A,#00Fh                 ;Low 4 bits only.
0138 F520            630             MOV     LastXY,A                ;Init LastXY
                     631
                     632     ;************
                     633     ;Set up to transmit self test report
                     634     ;************
013A C21B            635             CLR     RxEnable                ;Disable receiving.
013C 1194            636     SetUp:  ACALL   InitI2C
                     637
013E 750601          638     Report: MOV     I2CCxt,#RXIDLE          ;Set context idle receiver.     ;*RC*
0141 750BE0          639             MOV     SndType,#I_Attn
0144 D2DE            640             SETB    I2CFG.MASTRQB           ;Request to be master.
0146 20DEFD          641             JB      I2CFG.MASTRQB,$         ;Wait for message sent.         ;*PAS*
0149 E51A            642             MOV     A,SelfTest
014B 6005            643             JZ      RepDn                   ;Go if Selftest OK.
                     644
                     645             ;Selftest failed.
                     646             ;Send 2nd report and try to start anyway.
014D 751A00          647             MOV     SelfTest,#NO_ERROR
0150 80EC            648             SJMP    Report
                     649
0152 D21B            650      RepDn:  SETB    RxEnable                ;Enable Receiver and
                     651                                             ; fall through to MAIN.
                     652
                     653      $EJ
```

```
LOC  OBJ            LINE     SOURCE
                      654     ;*************************************************************************
                      655     ; Main Routine
                      656     ;
                      657     ; Sample X-Y quadrature inputs and compute mouse movement.
                      658     ; Accuracy requirement is:  +/- 3%  0-10 inches per second.
                      659     ; 10 inches per second @ 200 dpi means up to 2000 input changes/second.
                      660     ; Minimum (Nyquist) sampling rate is 4000/sec or sample every 250µs.
                      661     ; This is only two character times at 80k bps.
                      662     ; For best accuracy, should sample between every I2C character.
                      663     ;
                      664     ; Sample timing with 8 MHz crystal:
                      665     ;   no transition:      6 cycles        9µs
                      666     ;   X or Y transition:  25-31 cycles    38-47µs
                      667     ;   X and Y transition: 42-46 cycles    63-69µs
                      668     ;
                      669     ;*************************************************************************
                      670
0154                  671     MAIN:
0154 E58A             672             MOV     A,TL            ;Read timer to wait at least
0156 951C             673             SUBB    A,SampleClock   ; 64 cycles (96 µsec) between samples.
0158 30E6F9           674             JNB     ACC.6,MAIN
                      675             ;Take a sample
                      676             ;CLR    IE.EA           ;Protect sample code from interrupts.
015B 858A1C           677             MOV     SampleClock,TL
015E 3162             678             ACALL   Sample          ;Sample X-Y quadrature inputs.
                      679             ;SETB   IE.EA
0160 80F2             680             SJMP    MAIN
                      681
                      682     ; Sample X-Y quadrature inputs and update X-Y counters.
                      683     ;   b3-b0 = YB YA XB XA.
                      684     ;
                      685     ;   Channel A:   0 1 1 0 0 1 1 0 0     --->positive movement
                      686     ;   Channel B:   0 0 1 1 0 0 1 1 0     <---negative movement
                      687
                      688     ;   A and C are not preserved.
                      689     ;
0162 E5B0             690     Sample: MOV     A,P3            ;Read X & Y position detectors.
0164 540F             691             ANL     A,#00001111B    ;We only need the low 4 bits
0166 B52001           692             CJNE    A,LastXY,TRAN   ;Compare to last image.
0169 22               693             RET                     ; If no change, return.
                      694
016A 852022           695     TRAN:   MOV     TranXY,LastXY   ;Set up to calculate XY transition.
016D F520             696             MOV     LastXY,A        ;Save new P3 image.
016F 6222             697             XRL     TranXY,A        ;Mark transition bits (1=changed).
                      698
0171 20100A           699     XPULSE: JB      TranXY.0,XA     ;Branch on bit transitions.
0174 201113           700             JB      TranXY.1,XB
0177 201232           701     YPULSE: JB      TranXY.2,YA
017A 20133B           702             JB      TranXY.3,YB
017D 22               703             RET
                      704
                      705             ;Decode direction from quadrature.
                      706             ;Change in XA
017E E520             707     XA:     MOV     A,LastXY
0180 5403             708             ANL     A,#00000011B    ;Get X' X
0182 6010             709             JZ      XDEC            ;If 00, backward
0184 6403             710             XRL     A,#00000011B
0186 600C             711             JZ      XDEC            ;If 11, backward
0188 8016             712             SJMP    XINC            ;01 or 10, forward.
```

```
LOC  OBJ            LINE     SOURCE
                     713             ;Change in XB
018A E520            714     XB:     MOV     A,LastXY
018C 5403            715             ANL     A,#00000011B    ;Get X' X
018E 6010            716             JZ      XINC            ;If 00, forward
0190 6403            717             XRL     A,#00000011B
0192 600C            718             JZ      XINC            ;If 11, forward
                     719             ;01 or 10, backward, fall through to decrement
                     720
0194 7480            721     XDEC:   MOV     A,#080h         ;Do not decrement if
0196 6516            722             XRL     A,XCOUNT        ; count already at minimum –127.
0198 6004            723             JZ      XDEC2
019A 1516            724             DEC     XCOUNT
019C D21A            725             SETB    Movement        ;Note position has changed.
019E 80D7            726     XDEC2:  SJMP    YPULSE          ;Check for possible Y pulse.
                     727
01A0 747F            728     XINC:   MOV     A,#07Fh         ;Do not increment if
01A2 6516            729             XRL     A,XCOUNT        ; count already at maximum 127.
01A4 6004            730             JZ      XINC2
01A6 0516            731             INC     XCOUNT
01A8 D21A            732             SETB    Movement        ;Note position has changed.
01AA 80CB            733     XINC2:  SJMP    YPULSE          ;Check for possible Y pulse.
                     734
                     735             ;Change in YA
01AC E520            736     YA:     MOV     A,LastXY
01AE 540C            737             ANL     A,#00001100B    ;Get Y' Y
01B0 6010            738             JZ      YDEC            ;If 00, backward
01B2 640C            739             XRL     A,#00001100B
01B4 600C            740             JZ      YDEC            ;If 11, backward
01B6 8015            741             SJMP    YINC            ;01 or 10, forward.
                     742             ;Change in YB
01B8 E520            743     YB:     MOV     A,LastXY
01BA 540C            744             ANL     A,#00001100B    ;Get Y' Y
01BC 600F            745             JZ      YINC            ;If 00, forward
01BE 640C            746             XRL     A,#00001100B
01C0 600B            747             JZ      YINC            ;If 11, forward
                     748             ;01 or 10, backward, fall through to decrement.
                     749
01C2 7480            750     YDEC:   MOV     A,#080h         ;Do not decrement if
01C4 6517            751             XRL     A,YCOUNT        ; count already at minimum –127.
01C6 6004            752             JZ      YDEC2
01C8 1517            753             DEC     YCOUNT
01CA D21A            754             SETB    Movement        ;Note position has changed.
01CC 22              755     YDEC2:  RET
                     756
01CD 747F            757     YINC:   MOV     A,#07Fh         ;Do not increment if
01CF 6517            758             XRL     A,YCOUNT        ; count already at maximum 127.
01D1 6004            759             JZ      YINC2
01D3 0517            760             INC     YCOUNT
01D5 D21A            761             SETB    Movement        ;Note position has changed.
01D7 22              762     YINC2:  RET
                     763
                     764     $EJ
                     765                     ; I2C message processing contexts:
  0001               766     RXIDLE  EQU     1                 ;Idle receiver waiting for start.
  0002               767     RXBIT   EQU     2                 ;Waiting to receive a bit.
  0003               768     RXACK   EQU     3                 ;Waiting for ACK to complete.
                     769
  0004               770     TXBIT   EQU     4                 ;Waiting to send a bit.
  0005               771     TXREAD  EQU     5                 ;Waiting to read ACK.
```

```
LOC  OBJ            LINE     SOURCE
  0006               772     TXACK   EQU     6                ;Waiting for ACK.
                     773
                     774
                     775
                     776     ; I2C Interrupt Reasons
                     777     ;   Events CPU might be waiting for
                     778     ;     Receive
                     779     ;     (1) Start signal detected by idle slave (DRDY)
                     780     ;     (2) Next bit received (DRDY)
                     781     ;     (3) Acknowledge has been sent (DRDY)
                     782     ;     Transmit
                     783     ;     (4) Bus mastership granted (START and MASTER)
                     784     ;     (5) Ready to transmit next bit (DRDY)
                     785     ;     (6) Acknowledge received (DRDY)
                     786     ;   Unsolicited
                     787     ;   (7) Arbitration loss (ARL)
                     788     ;   (8) Sender aborted message (STOP)
                     789     ;   (9) Sender started new message before slave became idle (START)
                     790     ;
                     791     ; Only some of these events can occur at any time depending on
                     792     ; the state of sending or receiving a message.
                     793     ;
                     794     ; When the interrupt occurs, the keyboard needs to recover
                     795     ; the context from which it was sending or receiving a
                     796     ; message.  This context is maintained as follows:
                     797     ;
                     798     ;   I2Ccxt   I2C context, what event is expected.
                     799     ;   I2CDat   The byte being sent or received.
                     800     ;   BitCnt   Where we are in the sending or receiving the byte.
                     801     ;   ByteCnt  Where we are in sending or receiving a message
                     802     ;   Check    Computed checksum.
                     803     ;   RcvType  The message or command type being received.
                     804     ;              This may determine how successive bytes
                     805     ;              are to be processed.
                     806     ;   SndType  Type of message being sent or pending.
                     807     ;              This will determine how bytes are transmitted.
                     808     ;   SendRpt   Flag indicating CPU is waiting to send a Position
                     809     ;              report (and requested to become master).
                     810
                     811
                     812
                     813     $EJ
                     814     ;*********************
                     815     ; Enter I2C Interrupt
                     816     ;*********************
01D8 C2AC            817     I2CINT: CLR     IE.EI2                  ;disable the I2C interrupt
01DA 31F3            818             ACALL   XRETI                   ;then re-enable others
01DC C0D0            819             PUSH    PSW                     ;save registers
01DE C0E0            820             PUSH    ACC
                     821
                     822     ; Dispatch interrupt
01E0 309911          823     DISPAT: JNB     I2CON.MST,SLAVE
01E3 41B1            824             AJMP    MASTER                  ;go if we're master
                     825
                     826     ; Wait for ATN
01E5 7D08            827     WaitATN: MOV    R5,#DelayATN            ;Load ATN count (about 50mms)
01E7 209EF6          828     Wait1:  JB      I2CON.ATN,DISPAT        ;If ATN, dispatch next event,
01EA DDFB            829             DJNZ    R5,Wait1                ; else loop to try again.
                     830             ;If not seen after count tries,
```

```
LOC  OBJ            LINE     SOURCE
                     831             ; return from I2C interrupt.
                     832
                     833     ; Exit I2C interrupt
                     834     ; restore registers and return from interrupt
01EC D0E0            835     I2CRTI: POP     ACC
01EE D0D0            836             POP     PSW
01F0 D2AC            837             SETB    IE.EI2                  ;re-enable I2C interrupt
01F2 22              838             RET                             ;return to interrupted process
                     839
01F3 32              840     XRETI:  RETI                            ;used at start of service routine
                     841
                     842
                     843     $EJ
                     844     ;*************************
                     845     ; SLAVE RECEIVER
                     846     ;   R2=I2CDat, R3=BitCnt, R4=ByteCnt, R5=ATNCnt, R6=I2CCxt
                     847     ;*************************
                     848     ; Handle DRDY
                     849     ;*************************
01F4 309D5A          850     SLAVE:  JNB     I2CON.DRDY,NDRDY        ;Is it DRDY?
                     851     ; - context waiting for bit
01F7 BE0235          852             CJNE    R6,#RXBIT,NRxBit        ;Context waiting for bit?
01FA EA              853     Rx0:    MOV     A,R2                    ;Yes, get data in A
01FB DB21            854     Rx1:    DJNZ    R3,N8Bit                ;8th bit?
                     855
                     856             ; Read 8th bit
01FD A29F            857             MOV     C,I2CON.RDAT            ;Get 8th bit, don't clear ATN
01FF 33              858             RLC     A                       ;Include the 8th bit
0200 FA              859             MOV     R2,A                    ;Put data in R2.
0201 620D            860             XRL     Check,A                 ;XOR it to check
                     861
                     862             ;Send Acknowledge as appropriate.
0203 BC0007          863             CJNE    R4,#0,DoAck1            ;Address byte? (ByteCnt=0)
0206 B5080E          864             CJNE    A,MyAddr,NotMe          ;Is it my address?
0209 F50D            865             MOV     Check,A                 ;Yes, initialize check
020B C21F            866             CLR     AA                      ;AA=Flags.7
020D 852399          867     DoAck1: MOV     I2DAT,Flags             ;Assert Acknowledge (AA=Flags.7)
0210 7E03            868             MOV     R6,#RXACK               ;Set context waiting for ACK
0212 209D1D          869             JB      I2CON.DRDY,AckCmp       ;Can we skip waiting?
0215 21E5            870             AJMP    WaitATN                 ;Wait for ATN.
                     871     ; Not addressed to me
0217 7E01            872     NotMe:  MOV     R6,#RXIDLE              ;Set context to be idle receiver
0219 75987C          873             MOV     I2CON,#CDR+CSTR+CSTP+CARL+IDLE
021C 21EC            874             AJMP    I2CRTI                  ;Resume interrupted activity
                     875
                     876             ; Read bits 1-7
021E C2E7            877     N8Bit:  CLR     ACC.7
0220 4599            878             ORL     A,I2DAT                 ;Include the bit, clear ATN.
0222 23              879             RL      A                       ;Data comes in at MSB.
0223 209DD5          880             JB      I2CON.DRDY,Rx1          ;If DRDY, short cut.
0226 209DD2          881             JB      I2CON.DRDY,Rx1
0229 209DCF          882             JB      I2CON.DRDY,Rx1          ;One more try.
022C FA              883             MOV     R2,A                    ;Put data back.
022D 21E5            884             AJMP    WaitATN
                     885
                     886     ; - context waiting for ACK to complete
022F BE0311          887     NRxBit: CJNE    R6,#RXACK,NRxAck        ;Context waiting for ACK?
0232 7598A0          888     AckCmp: MOV     I2CON,#CDR+CXA          ;ACK complete, clr xmt.
                     889             ;Process complete byte.
```

```
LOC  OBJ            LINE     SOURCE
0235 618F            890             AJMP    DORXB
                     891             ; Return is AJMP DNRXB.
0237 0C              892     DNRXB:  INC     R4                      ;Increment ByteCnt.
                     893             ;ACALL  Sample
0238 7E02            894     SetRXB: MOV     R6,#RXBIT               ;Set context for next byte.
023A 7A00            895             MOV     R2,#0                   ;Clear receive buffer
023C 7B08            896             MOV     R3,#8                   ;BitCnt=8
023E 209DB9          897             JB      I2CON.DRDY,Rx0          ;If DRDY, short cut.
0241 21E5            898             AJMP    WaitATN                 ;Wait for ATN
                     899
                     900     ; – context idle slave
0243 BE010B          901     NRxAck: CJNE    R6,#RXIDLE,NRxIdle      ;Context idle slave?
0246 301BCE          902             JNB     RxEnable,NotMe          ;Am I enabled?
                     903             ;Yes, initialize to receive first byte
0249 7B07            904             MOV     R3,#7                   ;BitCnt=7 (remaining)
024B E4              905             CLR     A                       ;Data in A
024C FC              906             MOV     R4,A                    ;ByteCnt=0
024D 7E02            907             MOV     R6,#RXBIT               ;I2CCxt=receive next bit
024F 411E            908             AJMP    N8Bit
                     909     ; Context was not waiting for Next bit, ACK, or Idle slave.
                     910     ; Could be ARL.  Dispatch other flags.
                     911     ; Do not clear DRDY, we'll come back after ARL if necessary.
0251                 912     NRxIdle:
                     913
                     914
                     915     $EJ
                     916     ;*****************************************
                     917     ; It wasn't DRDY, could be ARL, START, or STOP
                     918     ; Handle ARL
                     919     ;*****************************************
0251 309C38          920     NDRDY:  JNB     I2CON.ARL,RxStop        ;Is it ARL?
                     921             ;Yes, note MASTRQ is still on unless we were sending STOP.
                     922             ;SndType is still pending.  If it was a position report,
                     923             ;indicate pending report in SendRpt in case SndType is needed.
0254 30DE2E          924     ARL0:   JNB     I2CFG.MASTRQB,Naddr     ;Was I sending STOP?
0257 E50B            925             MOV     A,SndType
0259 B40305          926             CJNE    A,#LD_Position,ARL01    ;Was I sending position?
025C 750B00          927             MOV     SndType,#I_NoMsg
025F D219            928             SETB    SendRpt
0261 B4E405          929     ARL01:  CJNE    A,#I_Error,ARL1         ;Was I sending Error?
0264 750B00          930             MOV     SndType,#I_NoMsg
0267 D220            931             SETB    MsgCheck
0269 301B19          932     ARL1:   JNB     RxEnable,NAddr          ;Am I enabled?
026C 209B39          933             JB      I2CON.STR,SlvStart      ;Handle start.
026F BC000A          934             CJNE    R4,#0,ARL3              ;Did we ARL in Address (ByteCnt=0)?
                     935     ; Lost arbitration in address, set context to read rest
                     936     ; of address in case message is to me.
0272 759810          937     ARL2:   MOV     I2CON,#CARL             ;Clear ARL.
0275 7E02            938             MOV     R6,#RXBIT               ;Set context waiting to read bit.
0277 209D80          939             JB      I2CON.DRDY,Rx0          ;If DRDY, short cut.
027A 21E5            940             AJMP    WaitATN
                     941     ; Lost arbitration outside dst addr
027C B4F006          942     ARL3:   CJNE    A,#I_Reset,NAddr        ;Was I sending I_Reset to my Addr?
027F C21C            943             CLR     TxSelfRst               ;Yes, reset flag since I lost.
0281 C21F            944             CLR     AA                      ;Acknowledge received bytes.
0283 80ED            945             SJMP    ARL2                    ;Message must be for me.
                     946     ; Message not for me, go back to idle receive.
0285                 947     NAddr:
0285 7598FC          948     IdleS:  MOV     I2CON,#CXA+IDLE+CARL+CDR+CSTR+CSTP
```

```
LOC  OBJ            LINE     SOURCE
0288 7E01            949             MOV     R6,#RXIDLE
028A 21EC            950             AJMP    I2CRTI
                     951     $EJ
                     952     ;*************
                     953     ; Handle STOP                           ;*RC*   Check for STOP first
                     954     ;*************
028C 309A16          955     RxStop: JNB     I2CON.STP,RxStart       ;Confirm it was Stop.
028F 759804          956             MOV     I2CON,#CSTP             ;Clear it.
0292 201F04          957             JB      AA,RxStop0              ;Check end of message reached,
                     958                                             ; Assert Acknowledge=1.
                     959             ; Received STOP before end of message.
0295 D220            960             SETB    MsgCheck                ;Signal interface error.
0297 D2DE            961             SETB    I2CFG.MASTRQB
0299 7E01            962     RxStop0: MOV    R6,#RXIDLE              ;Set context idle receiver.
029B 309E02          963             JNB     I2CON.ATN,RxStop1       ;If ATN, dispatch next event
029E 21E0            964             AJMP    DISPAT
02A0 759840          965     RxStop1: MOV    I2CON,#IDLE             ;Become idle.
02A3 21EC            966             AJMP    I2CRTI                  ;Resume interrupted activity
                     967
                     968     ;**************
                     969     ; Handle START
                     970     ;**************
02A5 309B07          971     RxStart: JNB    I2CON.STR,RxFault       ;Was it start?
02A8                 972     SlvStart:
02A8 759818          973             MOV     I2CON,#CSTR+CARL        ;Yes, clear it.
02AB 7C00            974             MOV     R4,#0                   ;ByteCnt=0
02AD 4138            975             AJMP    SetRXB                  ;Set up to receive byte.
                     976
                     977     ; It wasn't DRDY, ARL, START, or STOP.  Inconsistency error.
02AF 01B4            978     RxFault: AJMP   PwrUp
                     979
                     980
                     981     $EJ
                     982     ;***************************
                     983     ; MASTER TRANSMITTER
                     984     ;   R2=I2CDat, R3=BitCnt, R4=ByteCnt, R5=ATNCnt, R6=I2CCxt
                     985     ;***************************
                     986     ; Handle DRDY
                     987     ;***************************
02B1 EA              988     MASTER: MOV     A,R2                    ;Get data in A.
02B2 309D48          989             JNB     I2CON.DRDY,MNDRDY       ;Is it DRDY?
                     990     ; – context waiting to send bit
02B5 BE0418          991             CJNE    R6,#TXBIT,NTxBit        ;Context waiting to send bit?
02B8 F599            992     Tx1:    MOV     I2DAT,A                 ;Send bit
02BA 23              993     Tx2:    RL      A                       ;Rotate the byte.
02BB DB07            994             DJNZ    R3,MN8Bit               ;Was it 8th bit?
02BD 7E05            995             MOV     R6,#TXREAD              ;Set context waiting to read ACK
02BF 209D11          996             JB      I2CON.DRDY,TxAck1       ;If DRDY, short cut.
02C2 21E5            997             AJMP    WaitATN
                     998
                     999             ; prepare next bit 1–7
02C4 FA             1000     MN8Bit: MOV     R2,A                    ;Put the data back.
02C5 209DF0         1001             JB      I2CON.DRDY,Tx1          ;If DRDY, short cut.
02C8 209DED         1002             JB      I2CON.DRDY,Tx1
02CB 209DEA         1003             JB      I2CON.DRDY,Tx1          ;One more try.
02CE 21E5           1004             AJMP    WaitATN
                    1005
                    1006     ; – context waiting to read ACK
02D0 BE050D         1007     NTxBit: CJNE    R6,#TXREAD,NTxAck1      ;Context waiting to read ACK?
```

```
LOC  OBJ            LINE     SOURCE
02D3 7598A0         1008     TxAck1: MOV     I2CON,#CDR+CXA          ;Switch to receive mode.
02D6 7E06           1009             MOV     R6,#TXACK               ;Set context waiting for ACK.
02D8 209D08         1010             JB      I2CON.DRDY,NTxAck2      ;If DRDY, short cut.
02DB 23             1011             RL      A                       ;Align data in case ARL.
02DC 1B             1012             DEC     R3
02DD FA             1013             MOV     R2,A
02DE 21E5           1014             AJMP    WaitATN
                    1015
                    1016     ; – context waiting for ACK
02E0 BE0629         1017     NTxAck1: CJNE   R6,#TXACK,BeMast        ;Context waiting for ACK?
02E3 E598           1018     NTxAck2: MOV    A,I2CON                 ;Read from I2CON
02E5 5480           1019             ANL     A,#80h                  ;Only need 7th bit
02E7 6005           1020             JZ      AckOK
02E9                1021     BadAck: ;Stop if negative ACK.
02E9 D5097D         1022             DJNZ    NACnt,DoStp1            ;Keep pending msg till retry expires.
02EC 6166           1023             AJMP    DoStp
                    1024             ;Ack Okay, prepare to send next byte.
02EE 0C             1025     AckOK:  INC     R4                      ;Point to byte we want to send.
                    1026             ;ACALL  Sample
02EF 81B1           1027             AJMP    DOTXB
                    1028             ;return should AJMP DNTXB
02F1 EA             1029     DNTXB:  MOV     A,R2                    ;Next byte in A.
02F2 620D           1030             XRL     Check,A                 ;XOR with message check.
02F4 7B08           1031             MOV     R3,#8                   ;BitCnt=8
02F6 7E04           1032             MOV     R6,#TXBIT               ;Set context to send bit.
02F8 209DBD         1033             JB      I2CON.DRDY,Tx1          ;If DRDY, short cut.
02FB 21E5           1034             AJMP    WaitATN
                    1035
                    1036     $EJ
                    1037     ;*****************************************************
                    1038     ; It wasn't DRDY.  Could be ARL, START, or STOP.
                    1039     ; If ARL, align data with tx bit that lost arbitration
                    1040     ; go to handle as SLAVE (MASTER was cleared).
                    1041     ; If it was START, we just became master.
                    1042     ; If STOP, not sure how this occurred,
                    1043     ;   should have seen ARL, become idle SLAVE.
                    1044     ;*****************************************************
02FD 309C07         1045     MNDRDY: JNB     I2CON.ARL,MNARL         ;ARL?
0300 03             1046             RR      A                       ;Rotate data back.
0301 0B             1047             INC     R3                      ;Increment bit count.
0302 03             1048             RR      A
0303 0B             1049             INC     R3
0304 FA             1050             MOV     R2,A
0305 4154           1051             AJMP    ARL0
0307 209B02         1052     MNARL:  JB      I2CON.STR,BeMast
030A 4185           1053             AJMP    IdleS
                    1054
                    1055     ; – context START or not waiting for bit or ACK (from above).
                    1056     ; Must have just become master.  Start new message.
                    1057     ; If message is pending in send type, do it.
                    1058     ; Otherwise, check to send Position Report if needed.
030C A90B           1059     BEMAST: MOV     R1,SndType              ;Get message type.
030E B90043         1060             CJNE    R1,#I_NoMsg,SndMsg      ;No message pending?
0311 302007         1061             JNB     MsgCheck,BeMast1        ;Send error?
0314 750BE4         1062             MOV     SndType,#I_Error
0317 C220           1063             CLR     MsgCheck
0319 6154           1064             AJMP    SndMsg
031B 201902         1065     BeMast1: JB     SendRpt,PosMsg          ;Send position report?
031E 6166           1066             AJMP    DoStp                   ; then stop.
```

```
LOC  OBJ            LINE     SOURCE
                     1067
                     1068     ; Send Position report
0320 201C0C          1069     PosMsg: JB      TxSelfRst,PosMsg1       ;If first user data,
0323 D21C            1070             SETB    TxSelfRst
0325 750BF0          1071             MOV     SndType,#I_Reset        ;Send a I_Reset to my address.
0328 E508            1072             MOV     A,MyAddr
032A 750901          1073             MOV     NACnt,#1                ;No retry if not acknowledged.
032D 8027            1074             SJMP    SndMsg1
032F 750B03          1075     PosMsg1:MOV     SndType,#LD_Position    ;Send position report.
0332 C219            1076             CLR     SendRpt                 ;Clear pending report state.
0334 E4              1077             CLR     A                       ;Copy X-Y counts to xmt buffer.
0335 F512            1078             MOV     XBUF2,A                 ;Hi byte=0.
0337 F514            1079             MOV     YBUF2,A
0339 E516            1080             MOV     A,XCOUNT                ;Copy X.
033B F513            1081             MOV     XBUF1,A
033D 30E703          1082             JNB     ACC.7,Copy2             ;If negative,
0340 7512FF          1083             MOV     XBUF2,#0FFh             ; extend sign.
0343 E517            1084     Copy2:  MOV     A,YCOUNT                ;Copy Y.
0345 F515            1085             MOV     YBUF1,A
0347 30E703          1086             JNB     ACC.7,Copy3             ;If negative,
034A 7514FF          1087             MOV     YBUF2,#0FFh             ; extend sign.
034D E4              1088     Copy3:  CLR     A
034E F516            1089             MOV     XCOUNT,A                ;Reset X-Y counters.
0350 F517            1090             MOV     YCOUNT,A
0352 C21A            1091             CLR     Movement                ;Clear movement flag.
0354 7450            1092     SndMsg: MOV     A,#Adr_Host
0356 F599            1093     SndMsg1:MOV     I2DAT,A                 ;Send first bit by hand
0358 75981C          1094             MOV     I2CON,#CARL+CSTR+CSTP   ;Clear start, release SCL
                     1095             ;Set context for rest of address
035B F50D            1096             MOV     Check,A                 ;Init checksum
035D FA              1097             MOV     R2,A                    ;I2CDat=A
035E 7B08            1098             MOV     R3,#8                   ;BitCnt=8
0360 7C00            1099             MOV     R4,#0                   ;ByteCnt=0
0362 7E04            1100             MOV     R6,#TXBIT               ;Set context waiting to send bit
0364 41BA            1101             AJMP    Tx2
                     1102     $EJ
                     1103     ; Completed sending message, do STOP
0366 750B00          1104     DoStp:  MOV     SndType,#I_NoMsg        ;Indicate cmd no longer pending.
0369 717B            1105     DoStp1: ACALL   SndStop                 ;Send STOP.
036B E50B            1106             MOV     A,SndType               ;Is there a pending message?
036D 7006            1107             JNZ     DoStp3
036F 201903          1108             JB      SendRpt,DoStp3          ;Is there a Position Report?
0372 302004          1109             JNB     MsgCheck,DoStp4         ;Is there an error message?
0375 11AF            1110     DoStp3: ACALL   SDelay                  ;Yes, delay to give others
                     1111             ; a chance to become master without contention.
0377 D2DE            1112             SETB    I2CFG.MASTRQB           ;Request to be master again.
0379 4199            1113     DoStp4: AJMP    RxStop0                 ;Borrow code from receiver.
                     1114
                     1115     ; Send I2C STOP signal
037B C2DE            1116     SndStop:CLR     I2CFG.MASTRQB           ;Release Master request
037D 759821          1117             MOV     I2CON,#CDR+XSTP         ;Set to send stop
0380 309EFD          1118             JNB     I2CON.ATN,$             ;Wait for ATN
0383 759820          1119             MOV     I2CON,#CDR              ;Clear useless DRDY (rising SCL)
0386 309EFD          1120             JNB     I2CON.ATN,$             ;Wait for stop sent
0389 759894          1121             MOV     I2CON,#CARL+CSTP+CXA    ;Clear I2C bus
038C 7E01            1122             MOV     R6,#RXIDLE              ;Set context idle receiver.
038E 22              1123             RET
                     1124
                     1125     $EJ
```

```
LOC  OBJ            LINE     SOURCE
                    1126     ;*****************************
                    1127     ; DO_RX_BYTE
                    1128     ; Received a complete byte, already acknowledged.
                    1129     ; Examine context to decide what to do with it.
                    1130     ;
                    1131     ; Enter: R2 (I2CDat) is byte received.
                    1132     ;        R4 is offset to byte just received.
                    1133     ; Exit:  Command parameters saved as needed.
                    1134     ;        Checksum verified.  Valid commands executed.
                    1135     ;        Return by AJMP DNRXB
                    1136     ;
                    1137     ;   R2=I2CDat, R3=BitCnt, R4=ByteCnt, R5=ATNCnt, R6=I2CCxt
                    1138     ;*****************************
038F BC0002         1139     DORXB:  CJNE    R4,#0,DoRx1     ;Is it Address? (ByteCnt=0)?
0392 4137           1140             AJMP    DNRXB
0394 BC0102         1141     DoRx1:  CJNE    R4,#1,DoRx2     ;Is it source Addr? (ByteCnt=1)
0397 4137           1142             AJMP    DNRXB
                    1143     ;
0399 EA             1144     DoRx2:  MOV     A,R2            ;Get byte.
039A BC020C         1145             CJNE    R4,#2,DoRx3     ;Is it P+len?
039D 33             1146             RLC     A               ;Rotate Prot bit into C.
039E 9218           1147             MOV     Prot,C          ;Save it.
03A0 EA             1148             MOV     A,R2            ;Get ”len”.
03A1 547F           1149             ANL     A,#07Fh
03A3 2403           1150             ADD     A,#3            ;Add overhead.
03A5 F50C           1151             MOV     MsgLen,A        ;Save message length.
03A7 4137           1152             AJMP    DNRXB
                    1153     ;
03A9 BC0304         1154     DoRx3:  CJNE    R4,#3,DoRx4     ;Is it Command byte?
03AC F50A           1155             MOV     RcvType,A       ;Save it
03AE 4137           1156             AJMP    DNRXB
                    1157     $EJ
                    1158     ;
                    1159     ; Test for end of command
                    1160     ;   If command has no data, byte offset 4 will be the checksum.
03B0 E50C           1161     DoRx4:  MOV     A,MsgLen        ;Get message length.
03B2 B5040A         1162             CJNE    A,ByteCnt,ToMny ;End of command?
                    1163                                     ; sets carry if MsgLen<ByteCnt
03B5 D21F           1164             SETB    AA              ;Yes, do not Acknowledge more bytes.
03B7 E50D           1165             MOV     A,Check         ;Check in A
03B9 6002           1166             JZ      CheckOk         ;Bad check?
03BB 811F           1167             AJMP    RxErr
                    1168     ; Message check is Ok, dispatch valid commands
03BD 8125           1169     CheckOk: AJMP   DORXCMD         ;Return AJMP DNRXB
                    1170     ;
                    1171     ; Test for ByteCnt beyond message length
03BF 5002           1172     ToMny:  JNC     DoDat           ;Too many bytes?
                    1173             ; Yes, just exit, negative acknowledge already sent.
03C1 4137           1174     DoRx9:  AJMP    DNRXB
                    1175
                    1176     ;
                    1177     ; Receive message data bytes
                    1178     ;   ByteCnt from 4 to (MsgLen–1)
                    1179     ;
                    1180     ; Branch on RcvType to decide what to do with each byte.
                    1181     ; Notice Reset, Identify, and Poll have no data.
                    1182     ;   Protocol: Assign New Address, Capabilities request
                    1183     ;
03C3 EA             1184     DoDat:  MOV     A,R2            ;Get data again since DoRx4 wiped it.
```

```
LOC  OBJ            LINE     SOURCE
03C4 AF0A           1185             MOV     R7,RcvType      ;Put RcvType in R7 so we can CJNE
03C6 3018F8         1186             JNB     Prot,DoRx9      ;Ignore device data stream.
                    1187             ;Process C/S command bytes.
                    1188
                    1189     $EJ
                    1190     ;*********************************************
                    1191     ; DORXB: Assign
                    1192     ;  Compare incoming bytes with ID string (ByteCnt=4–29).
                    1193     ;    If not equal, set not equal bit and ignore.
                    1194     ;  Compare ByteCnt 30–31 with random number.
                    1195     ;    If not equal, ignore (become idle receiver).
                    1196     ;  Save ByteCnt 32, the address as parameter (R0).
                    1197     ;*********************************************
                    1198
03C9 BFF225         1199     Do5:    CJNE    R7,#I_AsgnAdr,Do6 ;Assign command?
03CC BC040B         1200             CJNE    R4,#4,Asn2      ;Start of ID string (ByteCnt=4)?
03CF 7900           1201             MOV     R1,#0           ;Initialize R1 is index
                    1202
                    1203             ; BytCnt <= 29
03D1 E9             1204     Asn4:   MOV     A,R1            ;Get offset
03D2 B18C           1205             ACALL   GET_ID
03D4 09             1206     Asn5:   INC     R1              ;Increment for next
03D5 B50215         1207             CJNE    A,I2CDAT,AsnIg  ;Compare to received byte
03D8 4137           1208             AJMP    DNRXB           ;Ok so far
                    1209
03DA BC1E02         1210     Asn2:   CJNE    R4,#30,Asn3     ;ByteCnt=30?
03DD 790E           1211             MOV     R1,#RandH       ;Yes, set to read random#
03DF 40F0           1212     Asn3:   JC      ASN4            ;Jump if less than 30.
                    1213
                    1214             ; ByteCnt >= 30
03E1 BC2004         1215             CJNE    R4,#32,Asn6     ;ByteCnt=32?
03E4 A802           1216             MOV     R0,I2CDat       ;Save new address in R0.
03E6 4137           1217             AJMP    DNRXB
03E8 5003           1218     Asn6:   JNC     AsnIg           ;Jump if ByteCnt>32.
03EA E7             1219             MOV     A,@R1           ;Get byte of random #.
03EB 80E7           1220             SJMP    Asn5            ;Steal code from above.
                    1221
03ED D21E           1222     AsnIg:  SETB    NotMyID         ;It’s not for me
03EF 4137           1223             AJMP    DNRXB
                    1224
                    1225     ; Application Capabilities Request
03F1 BFF321         1226     Do6:    CJNE    R7,#I_CapReq,Do7        ;CapRequest?
03F4 BC0403         1227             CJNE    R4,#4,Cpr1              ;ByteCnt=4?
03F7 EA             1228             MOV     A,R2                    ;Check Cap hi pointer=0?
03F8 7014           1229             JNZ     Cpr4                    ;No, reset length and offset to zero.
03FA BC0516         1230     Cpr1:   CJNE    R4,#5,Cpr9              ;ByteCnt=5?
                    1231             ;Yes, check for valid offset.
03FD E518           1232             MOV     A,CapOffset             ;Compute next offset.
03FF 2519           1233             ADD     A,CapLen
0401 B50202         1234             CJNE    A,I2CDat,Cpr2           ;Send next?
0404 8111           1235             AJMP    Cpr8                    ; Yes, jump to set offset.
0406 EA             1236     Cpr2:   MOV     A,R2                    ;A=received offset.
0407 6008           1237             JZ      Cpr8                    ;Send first?
0409 B51802         1238             CJNE    A,CapOffset,Cpr4        ;Send previous?
040C 4137           1239             AJMP    DNRXB                   ; Yes, use current offset.
040E E4             1240     Cpr4:   CLR     A                       ;Reset Length and offset to zero.
040F F519           1241             MOV     CapLen,A
0411 F518           1242     Cpr8:   MOV     CapOffset,A             ;Load Cap offset.
0413 4137           1243     Cpr9:   AJMP    DNRXB
```

```
LOC  OBJ            LINE     SOURCE
                    1244
                    1245     ; Set reporting Interval
0415 BF8205         1246     Do7:    CJNE    R7,#LD_SetInterval,Do7a
0418 BC0402         1247             CJNE    R4,#4,Do7a              ;ByteCnt=4?
041B A802           1248             MOV     R0,I2CDat               ;Save parameter in R0.
041D 4137           1249     Do7a:   AJMP    DNRXB
                    1250
                    1251     ; Command error (I_Error)
041F D220           1252     RxErr:  SETB    MsgCheck                ;Set to report error.
0421 D2DE           1253             SETB    I2CFG.MASTRQB
0423 4137           1254             AJMP    DNRXB
                    1255     $EJ
                    1256     ;************************
                    1257     ; DO_RX_CMD
                    1258     ; Valid command received, do it.
                    1259     ; Dispatch commands based on RcvType
                    1260     ; Parameter value is in R0.
                    1261     ;
                    1262     ; Commands Recognized:  I_Reset, I_IdReq, I_AsgnAdr, I_CapReq,
                    1263     ;                       App_Test, App_Poll, App_SetInterval
                    1264     ;
                    1265     ; Return is:  AJMP DNRXB.  A is not preserved.
                    1266     ;***************************
0425 AF0A           1267     DORXCMD:MOV     R7,RcvType              ;Put RcvType in R7 so we can CJNE.
0427 201802         1268             JB      Prot,DoC4               ;Go for Control/Status commands.
042A 4137           1269             AJMP    DNRXB                   ;Ignore device data stream msgs.
                    1270
                    1271     ;Bus protocol commands
                    1272     ; Reset
042C BFF002         1273     DoC4:   CJNE    R7,#I_Reset,DoC5
042F 01B4           1274             AJMP    PwrUp                   ;Do power up reset.
                    1275     ; Identify
0431 BFF112         1276     DoC5:   CJNE    R7,#I_IdReq,DoC6
0434 750BE1         1277             MOV     SndType,#I_IdReply      ;Message type is identify
0437 201D08         1278             JB      KeepID,RTBM             ;Keep same device number?
043A D21D           1279             SETB    KeepID
043C 858A0F         1280             MOV     RandL,TL                ;Random number <- T0
043F 858C0E         1281             MOV     RandH,TH
0442 D2DE           1282     RTBM:   SETB    I2CFG.MASTRQB           ;Request to be master
0444 4137           1283             AJMP    DNRXB
                    1284     ; Assign
0446 BFF20C         1285     DoC6:   CJNE    R7,#I_AsgnAdr,DoC7
0449 101E07         1286             JBC     NotMyID,NDo1            ;Was it a complete match?
044C BC21D0         1287             CJNE    R4,#33,RxErr            ;Check len=30+3
044F C21C           1288             CLR     TxSelfRst               ;Anticipate first user data.
0451 8808           1289             MOV     MyAddr,R0               ;Load new address
0453 4137           1290     Ndo1:   AJMP    DNRXB
                    1291     ; Capabilities Request
0455 BFF30C         1292     DoC7:   CJNE    R7,#I_CapReq,DoC8
0458 BC0600         1293             CJNE    R4,#6,$+3               ;Check len>=3+3
045B 40C2           1294             JC      RxErr
045D 750BE3         1295             MOV     SndType,#I_CapReply     ;Message type is Cap Report
0460 D2DE           1296             SETB    I2CFG.MASTRQB           ;Request to be master
0462 4137           1297             AJMP    DNRXB
                    1298     ; App Test
0464 BFB107         1299     DoC8:   CJNE    R7,#App_Test,DoC9
0467 750BA1         1300             MOV     SndType,#App_TestReply  ;Send a test report
046A D2DE           1301             SETB    I2CFG.MASTRQB           ;Request to be master.
046C 4137           1302             AJMP    DNRXB
```

```
LOC  OBJ            LINE     SOURCE
                     1303     ; App Poll
046E BFB00D          1304     DoC9:   CJNE    R7,#LD_Poll,DoC10
0471 E508            1305             MOV     A,MyAddr                ;Check for default address.
0473 B46E02          1306             CJNE    A,#Adr_Default,DoC9a
0476 8004            1307             SJMP    DoC9b                   ;Don't send Position reports
                     1308                                             ; to default address.
0478 D219            1309     DoC9a:  SETB    SendRpt                 ;Flag to send Position report
047A D2DE            1310             SETB    I2CFG.MASTRQB           ;Request to be master.
047C 4137            1311     DoC9b:  AJMP    DNRXB
                     1312
                     1313     ; Set Locator report interval
047E BF822E          1314     DoC10:  CJNE    R7,#LD_SetInterval,DoC11
0481 BC0500          1315             CJNE    R4,#5,$+3               ;Check len>=2+3
0484 4099            1316             JC      RxErr
0486 B80005          1317             CJNE    R0,#0,DoC10a            ;Check parameter range.
                     1318             ; parameter=0, polling only.
0489 758800          1319             MOV     TCON,#0                 ;Turn off Timer0.
048C 4137            1320             AJMP    DNRXB
048E B80800          1321     DoC10a: CJNE    R0,#8,$+3
0491 401A            1322             JC      DoC10c                  ;Jump if R0<8.
0493 B81A00          1323             CJNE    R0,#26,$+3
0496 5015            1324             JNC     DoC10c                  ;Jump if R0>=26.
                     1325             ; 8 <= param <= 25, compute Timer0 reload value.
0498 74FF            1326             MOV     A,#0FFh                 ;Start at FFFFh
049A F9              1327             MOV     R1,A
049B C3              1328     DoC10b: CLR     C                       ;Loop to subtract
049C 949A            1329             SUBB    A,#MSECL                ; R0 milliseconds.
049E C9              1330             XCH     A,R1
049F 9402            1331             SUBB    A,#MSECH
04A1 C9              1332             XCH     A,R1
04A2 D8F7            1333             DJNZ    R0,DoC10b
04A4 F58B            1334             MOV     RTL,A                   ;Set Timer0 reload.
04A6 898D            1335             MOV     RTH,R1
04A8 758810          1336             MOV     TCON,#INIT_TCON         ;Turn on Timer0.
04AB 4137            1337             AJMP    DNRXB
04AD 811F            1338     DoC10c: AJMP    RxErr
                     1339
                     1340     ; Unrecognized command (ignore it)
04AF 4137            1341     DoC11:  AJMP    DNRXB
                     1342
                     1343
                     1344     $EJ
                     1345     ;******************
                     1346     ; DOTXB
                     1347     ; Transmitted a complete byte, has been acknowledged.
                     1348     ; Get next byte based on context.
                     1349     ; Enter:
                     1350     ;   R4 (ByteCnt) is the offset of the byte we wish to send.
                     1351     ; Exit:
                     1352     ;   R2 (I2CDat) is the next byte to transmit (if any).
                     1353     ;   A is not preserved.
                     1354     ;   Return via AJMP DNTXB.
                     1355     ;
                     1356     ;   R2=I2CDat, R3=BitCnt, R4=ByteCnt, R5=ATNCnt, R6=I2CCxt
                     1357     ;******************
                     1358     ; – Source address
04B1 BC0104          1359     DOTXB:  CJNE    R4,#1,DoTx1     ;ByteCnt=1?
04B4 AA08            1360             MOV     R2,MyAddr       ;Send source addr
04B6 41F1            1361             AJMP    DNTXB
```

```
LOC  OBJ           LINE     SOURCE
                    1362
                    1363     ; – Message length
04B8 BC0258         1364     DoTx1:  CJNE    R4,#2,DoTx2      ;ByteCnt=2?
04BB 7A82           1365             MOV     R2,#082h         ;Use 2 as default length,
                    1366                                      ; P=1, Control/Status msg.
04BD 750C05         1367             MOV     MsgLen,#5        ;Include overhead
                    1368             ; Compute length based on message type.
                    1369             ; If not Position Report, Identify, Output Error,
                    1370             ; or Reset, the length is 2.
04C0 AF0B           1371             MOV     R7,SndType
                    1372     ; Position report
04C2 BF0307         1373             CJNE    R7,#LD_Position,TxA ;Position report?
04C5 7A06           1374             MOV     R2,#6           ;Data length is 6 (P=0, data stream).
04C7 750C09         1375             MOV     MsgLen,#9       ;6 plus 3 overhead.
04CA 41F1           1376             AJMP    DNTXB
                    1377     ; Attention
04CC BFE00C         1378     TxA:    CJNE    R7,#I_Attn,TxI
04CF E51A           1379             MOV     A,SelfTest      ;Check for ROM error
04D1 B40105         1380             CJNE    A,#ROM_ERROR,TxA9
04D4 7A83           1381     Len3:   MOV     R2,#083h        ;Use Len=3 to include checksum.
04D6 750C06         1382             MOV     MsgLen,#6
04D9 41F1           1383     TxA9:   AJMP    DNTXB
                    1384     ; Identify Reply
04DB BFE107         1385     TxI:    CJNE    R7,#I_IdReply,TxC
04DE 7A9D           1386             MOV     R2,#(80h+29)    ;Length for Identify.
04E0 750C20         1387             MOV     MsgLen,#(29+3)  ;Add overhead for MsgLen
04E3 41F1           1388             AJMP    DNTXB
                    1389     ; Capabilities Report
04E5 BFE31C         1390     TxC:    CJNE    R7,#I_CapReply,TxE
04E8 7410           1391             MOV     A,#CapFragLen           ;Get default fragment length.
04EA F519           1392             MOV     CapLen,A                ;Save it.
04EC 2518           1393             ADD     A,CapOffset             ;Find end of fragment.
04EE C3             1394             CLR     C
04EF 9475           1395             SUBB    A,#(CAP_END–CAP_START)  ;Is it beyond end of Cap String?
04F1 4006           1396             JC      TxC3                    ;No, use default length.
04F3 F4             1397     TxC1:   CPL     A                       ;Yes, shorten as needed.
04F4 04             1398             INC     A
04F5 2519           1399             ADD     A,Caplen
04F7 F519           1400             MOV     CapLen,A
04F9 E519           1401     TxC3:   MOV     A,CapLen        ;Get fragment length.
04FB 2483           1402             ADD     A,#083h         ;Compute data length.
04FD FA             1403             MOV     R2,A            ;Prepare to send it.
04FE 2483           1404             ADD     A,#083h         ;Add 3 overhead for MsgLen (clear C/S).
0500 F50C           1405             MOV     MsgLen,A
0502 41F1           1406             AJMP    DNTXB
                    1407     ; Checksum or message framing error
0504 BFE402         1408     TxE:    CJNE    R7,#I_Error,TxR
0507 8003           1409             SJMP    TxR1
                    1410     ; Reset
0509 BFF005         1411     TxR:    CJNE    R7,#I_Reset,TxU
050C 7A81           1412     TxR1:   MOV     R2,#081h        ;Length for Reset (80+1).
050E 750C04         1413             MOV     MsgLen,#4       ;1 plus 3 overhead.
0511 41F1           1414     TxU:    AJMP    DNTXB
                    1415
                    1416     ; – Command code
0513 BC0309         1417     DoTx2:  CJNE    R4,#3,DoTxLast          ;ByteCnt=3?
0516 AA0B           1418             MOV     R2,SndType              ;Send command code
0518 BA0302         1419             CJNE    R2,#LD_Position,TCC1    ; unless it is position report.
051B AA10           1420             MOV     R2,ReportBuf            ;In that case send 1st byte of report.
```

```
LOC  OBJ            LINE     SOURCE
051D 41F1           1421     TCC1:   AJMP    DNTXB
                    1422     ;
                    1423     ; – Test for last byte of command (message check)
051F E50C           1424     DoTxLast: MOV   A,MsgLen
0521 B50404         1425             CJNE    A,ByteCnt,DoTxEnd ;Last byte of command?
                    1426                                       ; sets Carry if A<ByteCnt
0524 AA0D           1427             MOV     R2,Check          ;Yes, send check.
0526 41F1           1428             AJMP    DNTXB
                    1429     ;
                    1430     ; – Test for beyond last byte of command
0528 5005           1431     DoTxEnd: JNC    DoTx3             ;Beyond last byte (check)?
052A 750905         1432             MOV     NACnt,#5          ;Reset Negative Ack retry count.
052D 6166           1433             AJMP    DoStp             ;Send STOP.
                    1434
                    1435
                    1436     ; Transmit message data bytes
                    1437     ; ByteCnt from 3 to (length+2)
                    1438     ; Dispatch based on command type
                    1439
052F AF0B           1440     DoTx3:  MOV     R7,SndType
0531 BF0309         1441             CJNE    R7,#LD_Position,DoT3    ;Position report?
0534 E504           1442             MOV     A,ByteCnt               ;Yes, send next byte.
0536 240D           1443             ADD     A,#ReportBuf–3
0538 F9             1444             MOV     R1,A
0539 8702           1445             MOV     I2CDat,@R1              ;R2=I2CDat=@R1
053B 41F1           1446             AJMP    DNTXB
                    1447
                    1448     ;Attention report
053D BFE009         1449     DoT3:   CJNE    R7,#I_Attn,DoT4
0540 AA1A           1450             MOV     R2,SelfTest             ;Send Power–up selftest and attention
0542 BC0502         1451             CJNE    R4,#5,AT4               ;ByteCnt=5?
0545 AA1B           1452             MOV     R2,RomSum               ;Yes, send checksum byte.
0547 41F1           1453     AT4:    AJMP    DNTXB
                    1454     ;
                    1455     ;Application test report
0549 BFA104         1456     DoT4:   CJNE    R7,#App_TestReply,DoT5
054C AA1A           1457             MOV     R2,SelfTest             ;Send Selftest result
054E 41F1           1458             AJMP    DNTXB
                    1459
                    1460     ;*****************************
                    1461     ; Identify report
                    1462     ;  Send ID string for ByteCnt 4–29 (26 bytes, last two are FFh).
                    1463     ;  Send random number for ByteCnt 30 and 31.
                    1464     ;*****************************
0550 BFE11D         1465     DoT5:   CJNE    R7,#I_IdReply,DoT6
0553 BC0409         1466             CJNE    R4,#4,IDR2      ;First byte (ByteCnt=4)?
                    1467             ;yes, set up to send ID string
0556 7900           1468             MOV     R1,#0           ;R1 is index
                    1469
0558 E9             1470     IDR4:   MOV     A,R1            ;Get offset
0559 B18C           1471             ACALL   GET_ID
055B FA             1472     IDR5:   MOV     R2,A            ;Prepare to send it
055C 09             1473             INC     R1              ;Increment for next
055D 41F1           1474             AJMP    DNTXB
                    1475     ;
055F BC1E02         1476     IDR2:   CJNE    R4,#30,IDR3     ;ByteCnt=30?
0562 790E           1477             MOV     R1,#RandH       ;Set to send random #.
0564 40F2           1478     IDR3:   JC      IDR4            ;Jump if less than 30.
                    1479             ; BytCnt >= 30
```

```
LOC  OBJ            LINE     SOURCE
0566 BC2000         1480             CJNE    R4,#32,$+3
0569 5003           1481             JNC     IDR7            ;Jump if >= 32.
056B E7             1482             MOV     A,@R1           ;Get byte of random #.
056C 80ED           1483             SJMP    IDR5
056E 6166           1484     IDR7:   AJMP    DoStp           ;Error! Beyond last byte,
                    1485                                     ; STOP now.
                    1486     ;
                    1487     ;*****************************
                    1488     ; Capability report
                    1489     ;  Send next byte of capability string.
                    1490     ;  Uses R1 as index.
                    1491     ;*****************************
0570 BFE317         1492     DoT6:   CJNE    R7,#I_CapReply,DoT7
0573 BC0404         1493             CJNE    R4,#4,Cap1      ;First byte (ByteCnt=4)?
0576 7A00           1494             MOV     R2,#0           ;Send High byte of Offset (always 0).
0578 41F1           1495             AJMP    DNTXB
                    1496
057A BC0506         1497     Cap1:   CJNE    R4,#5,CAP2      ;Second byte (Offsetlo)?
057D AA18           1498             MOV     R2,CapOffset    ;Send Low byte of Offset.
057F A918           1499             MOV     R1,CapOffset    ;Initialize R1 to use as index.
0581 41F1           1500             AJMP    DNTXB
                    1501
0583 E9             1502     Cap2:   MOV     A,R1            ;Get Capabilities Character.
0584 B1A9           1503             ACALL   GET_CAP
0586 FA             1504             MOV     R2,A            ;Prepare to send it.
0587 09             1505             INC     R1              ;Increment for next Character.
0588 41F1           1506     Cap3:   AJMP    DNTXB
                    1507     ;
                    1508     ; Unknown: How can we not know what we're sending?
058A 41F1           1509     DoT7:   AJMP    DNTXB
                    1510
                    1511
                    1512     $EJ
                    1513     ;**********************
                    1514     ; GET_ID
                    1515     ; Get byte of ID string
                    1516     ;   Enter: offset of desired byte in A.
                    1517     ;   Exit:  A is the desired byte.
                    1518     ;**********************
058C 04             1519     GET_ID: INC     A               ;Skip RET
058D 83             1520             MOVC    A,@A+PC         ;Get the byte
058E 22             1521             RET
                    1522     ;ID string is defined here, length is 25
058F 41             1523             DB      'A'             ;Protocol revision
0590 56312E31       1524             DB      'V1.1   '       ;Module revision
0594 202020
0597 44454320       1525             DB      'DEC     '      ;Vendor name
059B 20202020
059F 56535858       1526             DB      'VSXXX-BB'      ;Module name
05A3 582D4242
05A7 FF             1527             DB      0FFh            ;1st byte of device #
05A8 FF             1528             DB      0FFh            ;2nd byte of device #
                    1529
                    1530     ;**********************
                    1531     ; GET_CAP
                    1532     ; Get byte of Capabilities string.
                    1533     ; This implementation supports up to 254 bytes only!
                    1534     ;   Enter: offset of desired byte in A.
                    1535     ;   Exit:  A is the desired byte.
                    1536     ;**********************
```

```
LOC  OBJ            LINE     SOURCE
05A9 04             1537     GET_CAP: INC    A               ;Skip RET
05AA 83             1538              MOVC   A,@A+PC         ;Get the byte
05AB 22             1539              RET
                    1540     ;Capabilities string is defined here.
05AC 28             1541     CAP_START: DB   '('
05AD 2070726F       1542              DB     ' prot(locator)'
05B1 74286C6F
05B5 6361746F
05B9 7229
05BB 20747970       1543              DB     ' type(mouse)'
05BF 65286D6F
05C3 75736529
05C7 20627574       1544              DB     ' buttons(1(L)2(R)3(M))'
05CB 746F6E73
05CF 2831284C
05D3 29322852
05D7 2933284D
05DB 2929
05DD 2064696D       1545              DB     ' dim(2)'
05E1 283229
05E4 2072656C       1546              DB     ' rel'
05E8 20726573       1547              DB     ' res(200 inch)'
05EC 28323030
05F0 20696E63
05F4 6829
05F6 2072616E       1548              DB     ' range(-127 127)'
05FA 6765282D
05FE 31323720
0602 31323729
0606 20643028       1549              DB     ' d0(dname(X))'
060A 646E616D
060E 65285829
0612 29
0613 20643128       1550              DB     ' d1(dname(Y))'
0617 646E616D
061B 65285929
061F 29
0620 29             1551              DB     ')'
0621 00             1552     CAP_END: DB     0       ;Null terminator (not used).
                    1553     ; Capabilities length is 121 bytes
                    1554
                    1555              END
```