Apple® Desktop Bus (ADB™)

Author: *Rob McCall*
WFT Electronics
Support: *Gus Calabrese*
Dave Evink
Curt Apperson
WFT Electronics

INTRODUCTION

The purpose of this application note is to introduce a PIC16CXXX based ADB interface which can be used as a basis for the development of custom ADB devices. This application note describes; the hardware involved, a general purpose ADB protocol handler, and an example application task. The example software application supports a single key keyboard to the Macintosh® computer (Figure 1).

OVERVIEW

ADB licensing from Apple Computer.

Described as a peripheral bus used on almost all Macintoshes (except for the Macintosh 128, 512K, and Plus) for keyboards, mice, etc.

Communication between the ADB task and the application task takes place using several flags. The flags indicate whether there is data received that needs to be sent to the Macintosh, or if data from the Macintosh needs to be sent by the application.

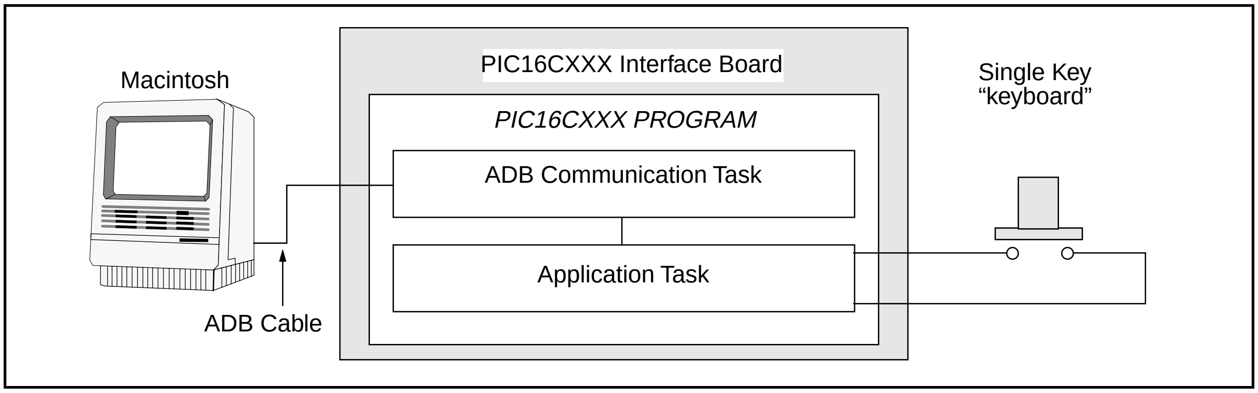
EXPLANATION OF ADB TECHNOLOGY

ADB is an asynchronous pulse-width communication protocol supporting a limited number of devices. All devices share a single I/O wire in a multi-drop master/slave configuration in which any slave device may request service. This is accomplished through a wired OR negative logic arrangement.

The ADB cable is composed of four wires: +5V, gnd, ADB signal, and power-on (of the Macintosh). The signal wire communicates ADB input and output using an open collector type signal. The number of devices is limited by the addressing scheme and a maximum current draw of 500 mA.

Every ADB device has a default address at start-up assigned by Apple. If there are device address conflicts, the protocol supports the reassignment of device addresses at start-up. The software in the PIC16CXXX discussed here is designed to easily modify the device address to make the PICmicro™ appear as another ADB device for testing and development.

FIGURE 1: BLOCK DIAGRAM OF FUNCTIONALITY



Macintosh, and ADB™ (Apple Desktop Bus) are trademarks/registered trademarks of Apple Computer, Inc.

No device issues commands, except the host. However, devices are permitted to request service during specific time intervals in the signal/Command protocol. A Service Request is referred to as an “Srq” The signal protocol communication is accomplished by pulling the ADB line low for various time intervals.

The host controls the flow of data through issuance of specific signal sequences and by issuing several types of Commands. The basic command types are Talk, Listen, Flush, and Reserved. Each command has a component called a “Register” indicator which specifies the storage area affected by the command type. The following is a summary explanation of the each of the commands. The complete specifications are available from Apple, as listed in the Resources section of this application note.

PROTOCOL ASSUMPTIONS

The ADB protocol is defined with a number of general assumptions about its use. These assumptions have driven the general philosophy of the communication sequences. It is assumed that the devices on the ADB are used for human input and each are used one at a time, such as a keyboard and a mouse. It is also assumed that the user’s transfer time from one device to another is relatively slow. This does not mean that the protocol is limited to these assumptions but rather that the protocol is optimized towards this type of use. This is made very evident in the host polling logic, where the host continues to poll the last device communicated with until another device issues an Srq. Consequently, if another device issues an Srq, the device being communicated with (or the host) may need to retransmit.

ADB Elements:

The ADB protocol has two components, a Signal protocol and a Command/Data protocol. These two elements are intertwined. The Signal protocol is differentiated in most cases by timing periods during which the ADB signal is low. The Apple ADB specification allows ± 3% tolerance timing of the signals from the host and ± 30% by the devices. The signals are:

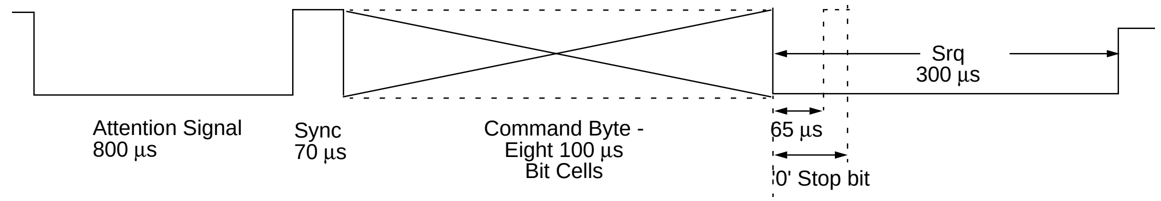
- Reset: signal low for 3 ms.
- Attention: signal low for 800 µs.
- Sync: signal high for 70 µs.
- Stop-to-Start-Time (`Tlt`): signal high for between 65 and 160 µs.
- Service Request (`Srq`): signal low for 300 µs.

After device initialization, in general, all communication through the ADB is accomplished through the following event sequence initiated by the host:

1. Attention signal
2. Sync signal
3. command packet
4. Tlt signal
5. data packet transfer

Depending upon the command, the device may or may not respond with a data packet. Service requests are issued by the devices during a very specific time at the end of the reception of the command packet.

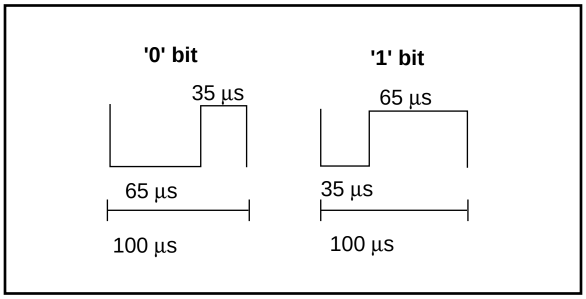
FIGURE 2: TYPICAL TRANSACTION WITH COMMAND AND DATA



If a device in need of service issues a service request, it must do so within the 65 µs of the Stop Bit’s low time and maintain the line low for a total of 300 µs.

The command packets and the data packets are the constructs used to communicate the digital information. The method of representing data bits is accomplished in a signal timing construct called a **bit cell**. Each **bit cell** is a 100 µs period. Data '1's and '0's are defined by the proportions of the bit cell time period when the line is low and then high. A '1' bit is represented by the line low for 35 µs, and high for 65 µs. Conversely, A '0' bit is represented by the line low for 65 µs, and high for 35 µs (Figure 3).

FIGURE 3: BIT CELLS



The **Command Packet**, received from the host, follows an Attention signal and a Sync signal. It consists of an 8-bit command byte and a '0' command stop bit. The command byte may be broken down into two nibbles. The upper nibble is a 4-bit unique device address. The lower nibble is defined as a Global or Reserved command for all devices, or a Talk, Listen, or Flush Command for a specific device. Also contained in the lower nibble is a "Register" designator which further details the Command. The importance of the Command Stop Bit Cell is that Srqs' can only be issued by a device to the host during the Command Stop Bit Cell low time if the device address is not for the device wishing service. The Host controls when Srq's are allowed through the Command protocol. The Tlt signal and Data Packet transfer, which are part of every Command packet signal sequence, are overridden if an Srq is issued by any device.

A **Data Packet** is the data sent to, or received from, the host. Its length is variable from 2 to 8 bytes. The structure is a '1' start bit, followed by 2 to 8 bytes, ending with a '0' stop bit. The Apple ADB documentation refers to the data packet sent or requested as Device Data "Registers". This does not necessarily indicate a specific place in memory. In this PIC16CXXX implementation, each Data Register has been limited to two PIC16CXXX register bytes. The ADB specification allows each Data Register to hold between two and eight bytes. They are referenced in the Command byte as "register" 0, 1, 2, or 3. Data Register 3 has special significance. It holds the special status information bits (such as whether Srq's are allowed), the Device Address, and the Device Handler ID. Commands are further defined by the "register id" sent in the Command data packet.

For example, if the Host issues the Command in binary of `0010 1100`, it would be interpreted as "Device 2, Talk Register 0". The complete definition of the Commands and data registers are described in detail in the ADB specifications supplied by Apple.

PIC16CXXX ADB PROTOCOL PROGRAM EVENT SEQUENCE

Overview

At power-on the host will generate a Reset signal. The purpose of Reset is to initialize the devices on the ADB line. This includes determining the addresses of each device, and resolving device address conflicts if there are any. Once the device addresses are determined, each device waits to be commanded or issues an Srq if it requires service from the host and is not being addressed by the host. After Reset processing, the ADB Protocol Task monitors the ADB line for the Attention/Sync/Command signal sequence. The PIC16CXXX program differentiates the signal timing.

Note: The signal detection routines check to see if the Application Task needs service after each event and after the falling edge of the Attention signal is detected.

Command interpretation is accomplished during the low signal time of the Stop Bit cell of the Command packet. Response to the Command must occur after the minimum time of the Stop to Start time period (Tlt), which is 160 µs. but before the max Tlt time of 240 µs. When a device has issued an Srq, it waits to be addressed by the host. If the next Command received is not for that device, it issues the Srq again. The normal response to an Srq will be a Talk Command from the host.

Detailed Description

Start-up

Upon start-up, the Reset routine is executed, looking for the ADB line to be high. When the line is high, an initialization routine is executed during which registers are cleared or loaded with default values. The only exception is a register for generating a random address used in the address conflict resolution process.

Reset

During a Reset condition, default values are loaded, such as the Default Device Address and Handler ID (a piece of information used by the host to identify the type of device). If more than one device has the same address, there is a sequence of events to resolve address conflicts described in the Implementation section. The host assigns a unique address to each device. The Reset condition only takes place once, during start-up, except under unusual conditions, such as testing this program.

Attention Routine

When the Reset routine is complete, the Attention Signal routine is executed, looking for the line to go low and then high. This low time is monitored to be within range of the Attention Signal Timing. If the timing is below the minimum threshold, the routine aborts to start over again looking for the line to go low at the beginning of the Attention Signal. If the low time is exceeded, the routine aborts to the Reset Signal routine.

Sync Signal Routine

When the line transitions to high, the Sync Signal routine looks for the line to go low at the start of the first bit of the Command Byte. If the Sync high time is exceeded, the routine aborts to the Attention Signal.

Command Routine

The Command routine detects and decodes the next 8 bit-cells as the Command Byte. The routine must first determine if the device address given is for itself. If the routine determines that the device address in the Command matches the stored device addresses, then it may do one of two things; issue an Srq to the host by holding the line low, or go on to check if the Command is Global to all devices. If the command is Global, the routine determines the specific Command and executes the routine for that Global Command. After execution of the Command routine it then goes back to look for the Attention Signal.

When a device is addressed, it determines whether the Command is to Talk, Listen, or Flush data, for the specified Data Register number. If the Command is for Data Register 3, there are special considerations described for this program in the Implementation section later in this application note. If the Command is to Flush, the routine clears the data in the specified register. The ADB specification defines the action of the Flush Command to be device specific. For a Talk Command or Listen Command, the device then waits for the Tlt signal. When the Command is to Talk, the device sends the data bytes from the specified register and a Data Stop Bit after the Tlt minimum time. For a Listen Command, the device receives data for the specified register.

When the data has been Flushed, Sent, or Received, the device then returns to monitoring for the Attention signal again.

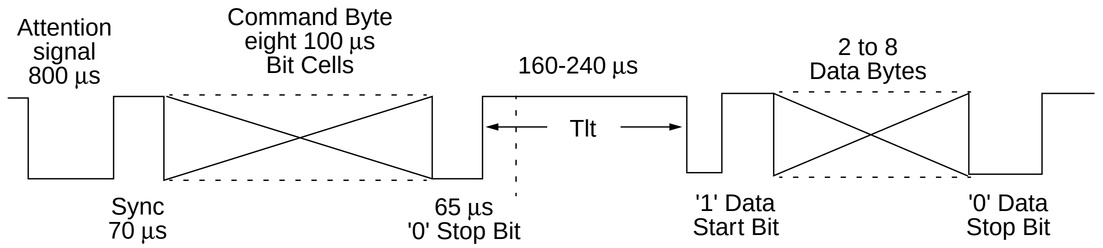
Note 1: In this PIC16CXXX program, the Application Task is serviced before looking for the Attention signal.

Note 2: If at any time the line is low or high outside of the timing ranges, the program aborts to check if an Attention or Reset signal has been issued by the Host. In the case of sending Data, the program goes first to the Collision routine.

Sending Data to the Host

Data is sent only in response to a Talk Command. For every data bit cell, the line is tested to go high at the proper time. If the line is still low, a collision has occurred. When a collision is detected, a collision flag is set, and the program aborts to look for a Command signal sequence.

FIGURE 4: TYPICAL TRANSACTION WITH SERVICE REQUEST



IMPLEMENTATION

Hardware

The hardware of this circuit is fairly simple. The circuit is powered via the +5V and GND wires of the ADB cable. The ADB I/O wire is connected to pin RA0 with a pull-up resistor to 5V. The T0CKI pin is tied to GND. The Master Clear ($\overline{\text{MCLR}}$) pin is tied to 5V.

This circuit uses a 4 MHz crystal as a timing reference, but higher values may be substituted. The software is designed to accommodate higher frequencies.

A pushbutton switch is used as the single key of the “keyboard.” One side is connected to port RB1 with a pull-up resistor to 5V, and the other side to GND. An LED is used to indicate that the ‘key’ has been pressed, with the positive side connected to pin RB0 and the negative side to GND.

Software

The program designated as “Application Tasks,” has two sections, one is setup to switch between a protocol support task for the ADB signal decode and processing, and the other section is the Application Task, in this case a single key “keyboard” routine. The ADB protocol task has priority. The first section of the code is the ADB protocol task, the second section is the Application Task, “Keyboard.” The two tasks communicate through flags which indicate that data needs to be sent, or that data has been received.

The Keyboard Task is run at two times; 1) during the Attention Signal, 2) between the end of the Data Stop Bit and the beginning of the Attention Signal. The Keyboard Tasks is given up to 500 µs during the Attention Signal, and 900 µs during the time between the end of the Data Stop Bit and the beginning of the Attention Signal. It is important to note here that the other tasks MUST NOT AFFECT TMR0 or the ADB time variable that the Attention Signal is using to keep track of the RTCC.

Timing

Timing is accomplished by first loading a constant into a time variable. This constant represents the maximum limit for the current routine, which may not necessarily be the maximum timing range for the current Signal. The TMR0 value is loaded into the working register, and subtracted from the time variable. The Carry bit of the STATUS register is tested to see if it is set or clear. If the bit is clear, the current timing limit has been exceeded. Further action is taken based on this status. It is important to keep the constant away from 255, or rollover may occur, giving inaccurate results. The prescaler is applied to the TMR0 as necessary.

The following are the timing ranges used by this program for ADB signals:

Reset	> 824 µs
Attention	776-824 µs
Sync	72 µs
Bit Cell	Up to 104 µs
'1' Bit low time	< 50 µs
'0' Bit low time	> 50 < 72 µs
Stop bit	0 Bit
Stop to Start (Tlt)	140-260 µs
Service Request (Srq)	300 µs

Note: The range of values given for 0 Bit, 1 Bit and Tlt timing are slightly wider than those given in the ADB specification.

How Address Conflicts are Resolved

During the start-up process the host sends a “Talk Register 3” command to each device address, and waits for a response. When a device recognizes that the Host issued a “Talk Register 3” command, it responds by sending a random address. During the transfer of each Bit Cell of the random address the signal line is monitored for the expected signal level. If the signal is not what is expected there is an address conflict. If the address is sent successfully, the host will respond with a Listen Command to that device. The command will have a new Device Address to which that device will move. The device then only responds to commands at the new address.

If there is a conflict, where two devices have the same default address, and respond at the same time, the device that finds the line low when it expects it to be high, immediately stops transmitting because it has determined that a collision has occurred. The device which detected the collision marks its address as unmovable and therefore ignores the address move Command, a Listen Register 3 Command. The device maintains the unmovable address condition until it has executed a successful response to the Talk Register 3 Command.

The host continues sending a Talk Register 3 Command at the same address until there is a time-out and no device responds. This is how conflicts are resolved when more than one device has the same address; for example, if two keyboards are connected.

Program Sequence:

Words in parenthesis, ( ), accompanying the TITLES are Labels of procedures in the corresponding code.

Start-up / IDLE (`Start`)

Start by configuring the ADB pin on PORTA and the Switch Pin on PORTB as inputs, and tri-stating the rest of PORTA and PORTB as outputs.

INITIALIZE DEFAULT VALUES WHEN THE LINE IS HIGH (`Reset`)

Look for the line to be high, and when it is, clear or initialize registers to default values.

LOOK FOR ATTENTION OR RESET (`AttnSig`)

Look for the line to go low, when it does, clear TMR0 and time how long it is low. An Attention Signal has occurred if the line went high between 776 and 824 µs. If the low time is measured to less than 776 µs, another signal has occurred and the program aborts, looking for the Attention Signal again. When the low time is measured to greater than 824 µs, the program interprets this timing as a Reset Signal. The program starts over again, waiting for the line to be high, and when it is, performs a Reset initialization.

Note:	The keyboard task is performed during the Attention Signal (Task_2).

LOOK FOR SYNC SIGNAL (`SyncSig`; calls `Srq`)

The Sync Signal is the high time between the rising edge of the Attention Signal and the falling edge of the first bit of the Command.

GET THE COMMAND (Command; calls `Get_Bit`)

Look for the Command; a combination of eight '0' and '1' bits. The MSb is sent first. This is achieved by calling the `Get_Bit` routine, which checks whether the maximum Bit Cell time is exceeded, if not, it looks for the rising edge at the end of the bit. When the bit is received, it is rotated into a variable, and the end of the bit cell is expected. When the falling edge of the next bit is detected, the routine clears TMR0 and returns to Command, which calls `Get_Bit` again until all 8-bits of the Command have been received.

ISSUE A SERVICE REQUEST IF NECESSARY (`Srq`)

If data needs to be sent to the Host, a Service Request (`Srq`) is issued by holding the line low while the Stop Bit is being received during the Stop-to-Start time (`Tlt`) which is between the end of the Command Stop Bit and the beginning of the Data Start Bit.

LOOK FOR STOP BIT (`CmdStop`)

Look for the Stop Bit (a '0' bit of 65 µs) that comes after the last Command Byte.

INTERPRET THE COMMAND (`AddrChk`)

After the command has been received, determine if the address belongs to this device. If the address is not for this device, determine if the command is global for all devices and if so, do that command. If this is not a Global/Reserved Command, call the Service Request (Srq) Routine to see if an Srq should to be issued to the Host, and do so if necessary, then return to get the Attn Signal. If the Address is for this device determine whether it is a Talk, Listen, or Flush Command, and go to the specified Command routine.

SENDING DATA (Talk; calls `Tlt`)

If the command was interpreted to be a Talk Command addressed to this device, call the Stop-to-Start Time (Tlt) routine. When the `Tlt` routine has completed, determine if this is a Talk Register 3 Command. If so, return a Random Address as part of the two bytes sent to the Host. If this is not a Talk Register 3 Command, determine if data needs to be sent. If so, send the Data Start Bit (a ‘1’), two bytes of data from the indicated register, and a Stop Bit (a ‘0’). If not, abort to the Attention Signal. If at any time the transmission of Data is interrupted, abort to the Collision routine. Only after a complete transmission should the flags be cleared indicating a successful transmission.

Note:	The ADB Specification indicates data may be between two and eight bytes long. The limitations of the PIC16C54/55/56 parts allow only two bytes of data to be sent by this program due to limited register space. If more than two bytes of data must be sent, use the PIC16C57.

RECEIVING DATA (Listen; calls `Tlt`)

If the command was interpreted to be a Listen Command addressed to this device, call the Stop-to-Start Time (`Tlt`) routine. When the `Tlt` routine has completed, receive the rest of the Data Start Bit, 2 Data Bytes, and Data Stop Bit. When the data has been received, determine whether this is a Listen Register 3 Command. If this is a Listen Register 3 Command, interpret what the command is. If this is a conditional Address Change Command, determine if this Device’s Address is moveable at this time. If not, abort to the Attention Signal. If so, change the device to the new address and go run the Second Application Task. If this is not a Listen Register 3 Command, move the data into the specified register and go run the Second Application Task.

LOOK FOR THE STOP TO START TIME (`Tlt`)

After the Command and Stop Bit, the Talk or Listen routines call the Tlt routine. Tlt looks for the line to go low. If the line went low before the minimum Tlt Time, see if this is a Talk Command. If this is a Talk Command, abort to the Collision routine. If this is a Listen Command, abort to the Attention Signal.

If the minimum Tlt time passes and the line is high, see if the Talk routine called the Tlt, if so, go wait for until the middle of the Tlt, then return to the Talk routine to send the Data Start Bit, Data Bytes, and Stop Bit. If at any time the line goes low during the Tlt and the Talk routine called it, abort to the Collision routine.

If the Listen routine did call Tlt, look for the line to go low at the beginning of the Data Start Bit. When the line goes low, return for the rest of the Start Bit. If the line doesn't go low before the maximum Tlt time is up, abort to the Attention Signal.

THE KEYBOARD TASK IS PERFORMED BETWEEN THE END OF THE DATA STOP BIT AND THE ATTENTION SIGNAL (Task_2)

The Keyboard Task checks to see if the key has been pressed. When the key is pressed, indication flags are set and an LED is turned on until the key has been debounced. The flags allow the key to be debounced, Srq(s) to be sent to the Host, and indicate to the Talk routine that Data needs to be sent. Two bytes of data are loaded into Register 0 representing a key-down code and a flag is set indicating to the ADB task that data needs be sent to the host. When the key-down codes have been sent, the key-up codes are loaded into Register 0. When the key-up codes have been sent and the key has been debounced, the flags are cleared. The final routine of Task_2 decides whether to return to the beginning or middle of the Attention Signal.

FIGURE 5: APPLE DESKTOP BUS PIC16CXXX FLOWCHART

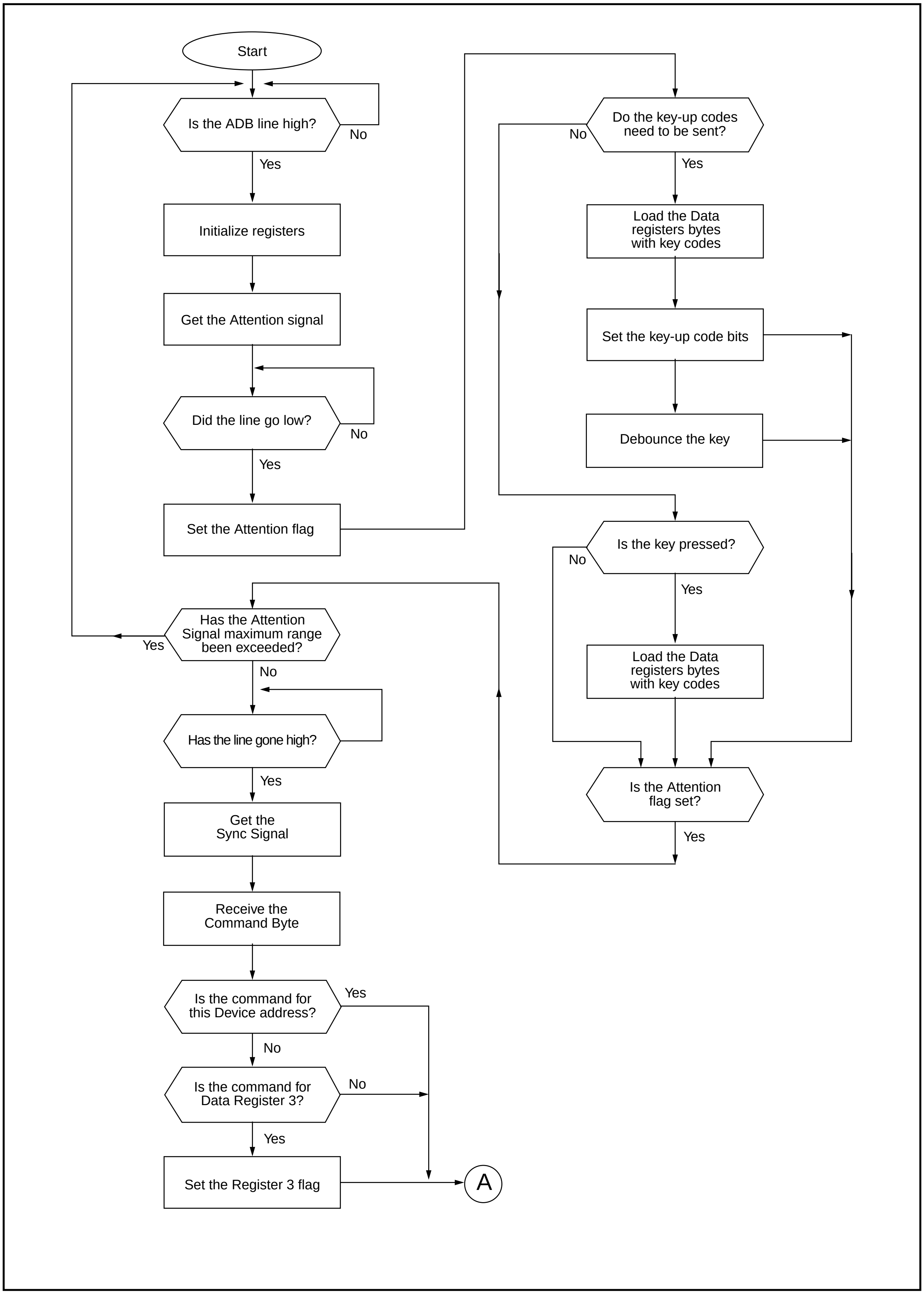
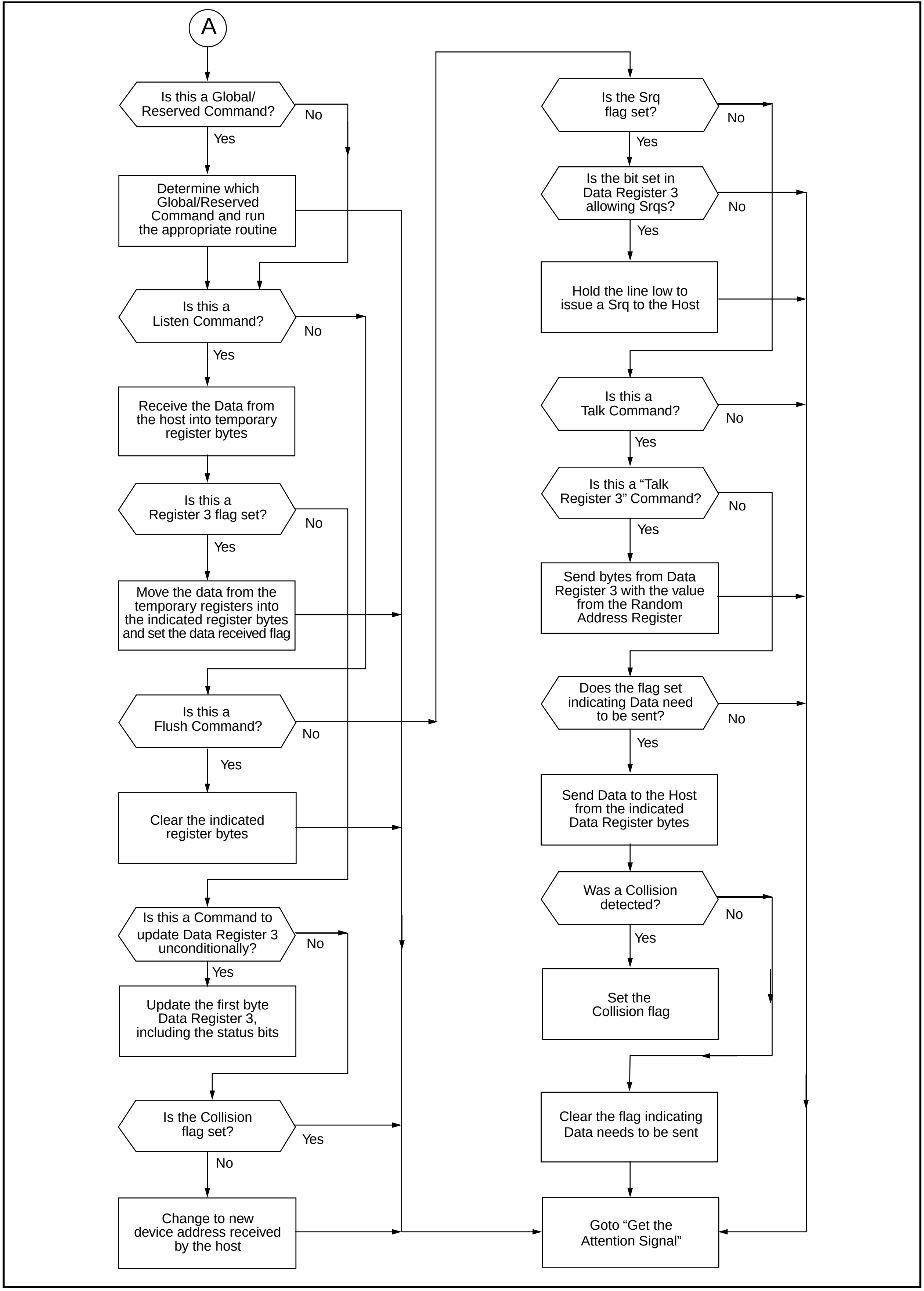


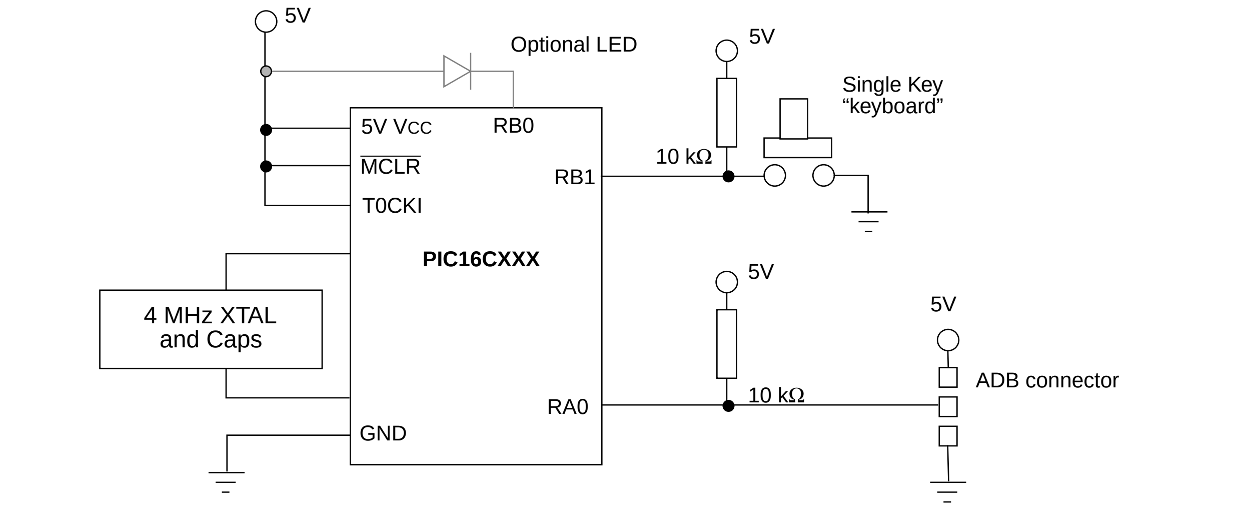
FIGURE 5 (CONT.): APPLE DESKTOP BUS PIC16CXXX FLOWCHART


A
Is this a Global/ Reserved Command?
No
Yes
Determine which Global/Reserved Command and run the appropriate routine
Is this a Listen Command?
No
Yes
Receive the Data from the host into temporary register bytes
Is this a Register 3 flag set?
No
Yes
Move the data from the temporary registers into the indicated register bytes and set the data received flag
Is this a Flush Command?
No
Yes
Clear the indicated register bytes
Is this a Command to update Data Register 3 unconditionally?
No
Yes
Update the first byte Data Register 3, including the status bits
Is the Collision flag set?
Yes
No
Change to new device address received by the host
Is the Srq flag set?
No
Yes
Is the bit set in Data Register 3 allowing Srqs?
No
Yes
Hold the line low to issue a Srq to the Host
Is this a Talk Command?
No
Yes
Is this a "Talk Register 3" Command?
No
Yes
Send bytes from Data Register 3 with the value from the Random Address Register
Does the flag set indicating Data need to be sent?
No
Yes
Send Data to the Host from the indicated Data Register bytes
Was a Collision detected?
No
Yes
Set the Collision flag
Clear the flag indicating Data needs to be sent
Goto "Get the Attention Signal"

SUGGESTIONS ABOUT MODIFYING THE CODE

1. If high crystal frequencies are used, a divider equate (equ) at the beginning of the timing section of the equates allows an easy adaptation for all established timing definitions.
2. The second application task may occur as a communication task with another PIC16CXXX device by using the three other I/O lines on PORTA, although test code for this has not yet been written. Two of the lines would be used as ready-to-send (one for each PIC16CXXX). The third would be used as a data line, using low signals as '0' bits, and high signals as '1' bits. Additionally, all eight lines on PORTB may be used as well.

FIGURE 6: SIMPLE SCHEMATIC OF THE TEST BOARD



RESOURCES

Apple Publications and Support Software

MacTech Magazine (formerly MacTutor) is a publication dedicated to supporting the Macintosh. They have had several articles regarding the Apple Desktop Bus. They publish a CD-ROM that contains all of their articles from 1984 to 1992. Also, single disks are available (ask for #42).

MacTech Magazine can be contacted at:

P.O. Box 250055
Los Angeles, CA 90025-9555
310 575-4343 FAX 310 575-0925
Applelink: MACTECHMAG
Internet: info@xplain.com

Apple licenses the ADB technology. They can be contacted at:

20525 Mariani Ave.
Cupertino, CA 95014
Attn: Software Licensing

- Apple Keyboard, extended, specification drawing #062-0168-A.
- Apple Desktop specification drawing # 062-0267-E.
- Apple Desktop connector, plug, Mini DIN drawing #519-032X-A.
- Engineering Specification, Macintosh transceiver interface, ADB drawing #062-2012-A.
- Apple keyboard, specification drawing #062-0169-A.
- Developer CD series, Tool Chest Edition, August 1993 contains:
  - Folder = Tool Chest: Devices and Hardware: Apple Desktop Bus
  - ADB Analyzer
  - ADB Parser (most complete environment)
  - ADB Lister
  - ADB ReInit
  - ADB Tablet code samples

WFT Electronics offers free assistance in procuring necessary ADB info. Contact Gus Calabrese, Rob McCall, Dave Evink at:

4555 E. 16th Ave.
Denver, CO 80220
303 321-1119 FAX 303-321-1119 Applelink: WFT
Internet: Gus_Calabrese@onenet-bbs.orgA

AUTHOR / CREDITS

Rob McCall developed the majority of the PIC16CXX ADB code. He also wrote most of the application note. Gus Calabrese, Dave Evink, and Curt Apperson supported this effort. Dave works with Gus, Rob, and Curt in developing a variety of embedded processor products.

Contact Gus Calabrese, Rob McCall, Dave Evink, Curt Apperson at:

WFT Electronics
4555 E. 16th Ave.
Denver, CO 80220
303 321-1119 FAX 303-321-1119 Applelink: WFT
Internet: Gus_Calabrese@onenet-bbs.org

Please check the Microchip BBS for the latest version of the source code. Microchip’s Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: ADB.ASM

```
MPASM 01.40 Released              ADB.ASM   1-16-1997  17:26:35         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                    00001         LIST    P = 16C56, n = 66,   c=132,  E=0,  N=60
                    00002 ;
                    00003 ;*************************************************************A113-0004
                    00004 ;
                    00005 ; ADB.ASM *** This program is for PIC16C5x microcontrollers:
                    00006 ;
                    00007 ;       Program:          ADB.ASM
                    00008 ;       Revision Date:
                    00009 ;                         1-16-97      Compatibility with MPASMWIN 1.40
                    00010 ;
                    00011 ;**********************************************************************
                    00012 ;
                    00013 ;**TESTING - The purpose of this program is to emulate a keyboard that
                    00014 ;  is Apple Desktop Bus (ADB) based.  The program allows the PIC to
                    00015 ;  appear to the Macintosh computer as a keyboard with a single key.
                    00016 ;  The code isdesigned to easily modify the device address to make the
                    00017 ;  PIC appear as another ADB device,which has its own proprietary
                    00018 ;  functions.
                    00019 ;**********************************************************************
                    00020 ;
                    00021 ; OVERVIEW OF ENTIRE PROGRAM:
                    00022 ; This program is setup to switch between a communication task with the
                    00023 ; the Apple Desktop Bus (ADB), and another application task.
                    00024 ; The ADB communication task has priority.
                    00025 ; All communication with the ADB is done using a single i/o line to
                    00026 ; the PIC, line RA0 on Port A.
                    00027 ; The second application may occur as a communication task with
                    00028 ; another PIC  chip as follows:
                    00029 ; Communication with the second PIC may be achieved by using the three
                    00030 ; other i/o lines on Port A.  Two of the lines would be used as
                    00031 ; ready-to-send (one for each PIC).  The third would be used as a data
                    00032 ; line, using low signals as 0 bits, and high signals as 1 bits.
                    00033 ; Additionally, all eight lines on PORTB may be used as well.
                    00034 ;
                    00035 ;************************  ADB COMMUNICATION TASK  *******************
                    00036 ;
                    00037 ;**** A BRIEF DESCRIPTION OF THE ADB COMMUNICATION SEQUENCE:
                    00038 ;
                    00039 ;  STARTUP    ------- initialize the TMR0 prescaler & Tri-States PORTA
                    00040 ;
                    00041 ;  Look for the following signals and/or take appropriate actions:
                    00042 ;  RESET ----------- a high line, then initialize default register values
                    00043 ;  ATTENTION ------- Attention signal, (there is enough time during this
                    00044 ;                    signal to allow other tasks to be performed)
                    00045 ;  COMMAND --------- 8 Command bits followed by a Stop Bit
                    00046 ;  INTERPRET ------- Decide whether the Host is addressing this Device,
                    00047 ;                    if so, decide what Command the Host issued
                    00048 ;                    if not, see if the Command is global to all Devices,
                    00049 ;                    also determine if the other Application needs to
                    00050 ;                    issue a Service to the Host.
                    00051 ;  Tlt ------------ The time between the Stop bit of the Command byte and
                    00052 ;                    the Start Time of the data being received/sent. Also
                    00053 ;                    referred to as Stop to Start Time.
                    00054 ;  SERVICE REQUEST - in order for a Device to alert the Host that it has
```

```
;           data to send, the line is held down after the Command Stop
;           Bit (continuing on from the Tlt).
;   DATA -- a Data Start Bit, followed by 2 Data Bytes (up to as
;           many as 8 Bytes), and a final Stop Bit
;
*************************************************************************
;
; THE FOLLOWING IS A MORE DETAILED DESCRIPTION OF THE PROGRAM SEQUENCE:
;
; NOTE: words in parenthesis accompanying the TITLES are Labels of
; procedures corresponding in the code below.
;
;*** STARTUP / IDLE ***  (Start)  ***
; Startup by setting the ADB pin on PORTA as an input and tri-stating the
; rest as outputs. The routine then goes to the Reset routine.
; NOTE: For testing, pin RB1 is is set as an input, and the rest of PORTB
; is tri-stated as an output.
;
;*** INITIALIZE DEFAULT VALUES WHEN THE LINE IS HIGH ***  (Reset)  ***
; Look for the line to be high, and when it is, initializes the
; registers to default values.
;
;*** LOOK FOR ATTENTION OR RESET ***  (AttnSig)  ***
; Look for the line to go low, when it does, clear the TMR0 and time how
; long it’s low.
; An Attention Signal has occurred when the line goes high between 776 and
; 824 usecs.
; If the low time is measured less than 776 usecs, another signal has
; occurred and the program aborts, looking for the Attention Signal
; again. When the low time is measured greater than 824 usecs, the program
; interprets this timing as a Reset Signal.  The program starts over
; again, waiting for the line to be high, and when it is, performs a
; Reset initialization.
;*** OTHER APPLICATION TASKS MAY BE PERFORMED DURING
;                              THE ATTENTION SIGNAL *** (Task_2) ***
; The time during which the Attention signal takes place allows a second
; state to occur. The other task(s) is/are given up to 500 usecs during
; the  Attention Signal (900 usecs are given to the 2nd Task during the
; time between the end of the Data Stop Bit and the beginning of
; the Attention Signal.
; It is important to note here that the other task(s) MUST NOT AFFECT
; THE Timer0 or the time variable (TimeVar) that the Attention Signal is
; using to keep track of the TMR0.
;
;********** NOTE:
; If at any time during the detection of the Signals below, the line is
; low or high outside of timing ranges, the routine aborts to see if an
; Attention  or Reset signal has been issued by the Host, or, in the
; case of sending  Data, to the Collision routine.
;
;*** LOOK FOR SYNC SIGNAL ***  (SyncSig)  ***
; The Sync Signal is the high time between the rising edge of the
; Attention Signal and the falling edge of the first bit of the Command.
;
;*** GET THE COMMAND ***  (Command; calls GetBit)  ***
; Look for the Command, a combination of eight 0 and 1 bits, MSB sent
; first. This is achieved by calling a the GetBit routine which checks
; whether the maximum time is exceeded, if not, looks for the rising edge
; at the end of the bit.  When the bit is received, it is rotated into a
; variable, and the end of the bit cell is expected.  When the falling
; edge of the next bit is detected, the routine clears TMR0 and
; returns to Command, which calls GetBit again until all 8 bits of the
; Command have been received.
;*** ISSUE A SERVICE REQUEST IF NECESSARY ***  (Srq)  ***
; If data needs to be sent to the Host, issue a Service Request (Srq) by
; holding the line low while the Stop Bit is being recieved, during the
```

```
;Stop-to-Start time (Tlt) between the end of the Command Stop bit and
;the beginning of the Data Start Bit.
;
;*** LOOK FOR STOP BIT ***  (CmdStop)
;Look for the Stop Bit (a 0 bit of 65 usecs) that comes after the last
;Command Byte.
;
;*** INTERPRET THE COMMAND ***  (AddrChk)  ***
;After the Command has been received, determine if the Address belongs to
;this Device.
;If the Address is not for this Device determine if the command is
;global for all Devices and if so, do that command.
;If this is not a Global/Reserved command, call the Service Request (Srq)
;routine to see if an Srq should to be issued to the Host, and do so if
;necessary, then return to get the Attn Signal.
;If the Address is for this Device determine whether it is a Talk,
;Listen, or Flush Command, and go to the specified command routine.
;
;**IF COMMAND IS TALK OR LISTEN, LOOK FOR STOP TO START TIME ** (Tlt) **
;After the Command and Stop Bit (a 0 bit) the Talk or Listen routine
;calls the Tlt routine:
;look for the line to go low,
;if the line went low before the Min. Tlt Time, see if this is a Talk
;Command if this is a Talk Command, abort to the Collision routine
;if this is a Listen Command, abort to the Attention Signal
;if the Min. Tlt time passes & the line is high,
;see the Talk routine called the Tlt,
;if so, go wait for until the middle of the Tlt, then return to
;Talk to send the Data Start Bit, Data Bytes, and Stop Bit.
;if at any time the line goes low during the Tlt, abort to the
;Collision routine
;if Listen called the Tlt,
;look for the line to go low as the beginning of the Data Start Bit
;if the line goes low, return for the rest of the Start Bit
;if the line doesn’t go low before the Max. Tlt time,
;abort to the Attention Signal
;
;*** SENDING DATA ***  (Talk)  ***
;If the Command was interpreted to be a Talk Command addressed to this
;Device, call the Stop-to-Start Time (Tlt) routine.
;When the Tlt routine has completed, determine if this is a Talk Register
;3 Command.  If so, and if so, return a Random Address as part of the
;two bytes sent to the Host.
;if this is not a Talk Register 3 Command, determine if Data needs to be
;sent.  If so, send the Data Start Bit (a ‘1’), two bytes of Data,
;and a Stop Bit (a ‘0’).  If not, abort to the Attention Signal
;If at any time the transmission of Data is interrupted, abort to the
;Collision routine.  Only after a complete transmission should the
;flags be cleared indicating a successful transmission.
;NOTE:  The ADB Spec. indicates data may be between 2 and 8 bytes long.
;The limitations of the PIC 16C54/55/56 parts allow only 2 bytes of data
;to be sent by this program due to limited register space. If more than
;2 bytes of data must be sent, use the PIC16C57.
;
;*** RECEIVING DATA ***  (Listen)  ***
;If the Command was interpreted to be a Listen Command addressed to this
;Device, call the Stop-to-Start Time (Tlt) routine.
;When the Tlt routine has completed, receive the rest of the Data
;Start Bit, 2 Data Bytes, and Data Stop Bit.
;When the Data has been received, determine whether this is a Listen
;Register 3 Command.
;if this is a Listen Register 3 Command, interpret what the Command
;is.  If this is a conditional Address change command, determine if
;this Device’s Address is moveable at this time.  If not, abort to the
;Attention Signal.  If so, change the Device to the new Address and
;go run the Second Application Task.
```

```
            ; if this is not a Listen Register 3 Command, move the Data into the
            ; specified register and go run the Second Application Task.
            ;
            ;***********************************************************************
            ;
            ;*** TIMING ALGORITHM ***
            ; Timing for ADB signals is done by clearing the TMR0, loading a constant
            ; into  a time variable, subtracting the TMR0 from the variable,
            ; This process is looped until the either the Carry Bit in the Status
            ; Register is clear, indicating the amount of time in the time variable
            ; has elapsed, or the condition of the data line has been met.
            ; If the line goes high or low at an inappropriate time, an error has
            ; occurred, and the current operation should be aborted.
            ;
            ; NOTE: The minimum and maximum values given below for 0 bit, 1 bit, and
            ;       Tlt timing are slightly shorter and longer than those given in
            ;       the ADB specification.
            ;  The following are the timing ranges used
            ;  by this program for ADB signals:
            ;  Reset ........Greater Than 824  usecs
            ;  Attention..............776-824  usecs
            ;  Sync.......................72   usecs
            ;  Bit Cell.............Up to 104  usecs
            ;      1 Bit...............32-40   usecs
            ;      0 Bit...............60-72   usecs
            ;  Stop bit................60-72   usecs
            ;  Stop to Start (Tlt)....140-260  usecs
            ;  Service Request (Srq)......300  usecs
            ;
            ;
            ; A SOMEWHAT GRAPHICAL REPRESENTATION OF THE TIMING SIGNAL RANGES (in
            ; usecs):
            ;  |-------|--------|--------|
            ;        30-40    60-70     100
            ;        1 Bit   0 Bit    End of Bit Cell
            ;
            ;  |--------------|----------|
            ;              140-260      300
            ;                Tlt        Srq1
            ;
            ;  |-----------------------------------|--------|---|--------------->
            ;  0                                 776      824  Greater than 824....
            ;  Signal invalid in this area--------|  AttnSig    Reset
            ;
            ;***********************************************************************
            ;
            ;*******************   THE PROGRAM BEGINS HERE   *********************

                include “p16c5X.inc”   ; default EQUates for the PIC registers
                    LIST
            ; P16C5X.INC Standard Header File, Version 3.30 Microchip Technology
                    LIST

000001FF    PIC54   equ     1FFh   ; Define the Reset Vector for 16c54.

00000000    NULL    equ     00h    ; used for returning nothing from a called routine

00000000    LSB     equ     00h    ; Least Significant Bit
00000007    MSB     equ     07h    ; Most Significant Bit

00000000    FALSE   equ     00h    ; For Boolean tests
00000001    TRUE    equ     01h

                include “adb.equ”  ; ADB EQUates
            ;************************ ADB.EQU Header-sets up EQUates ************

```

```
                  00003 ;*** TESTING *** BITS USED IN TESTING FOR I/O
                  00004
                  00005 ; *** BOOLEANS USED TO SELECT PART BEING USED
                  00006                        ; Only One Part May Be selected at a time
00000000          00007 C54     equ     FALSE  ;TRUE
00000000          00008 C55     equ     FALSE
00000001          00009 C56     equ     TRUE   ;FALSE
00000000          00010 C57     equ     FALSE
                  00011
00000000          00012 LED     equ     00h    ; ***AN LED ON LINE RB0 INDICATES SWITCH PRESSED
00000001          00013 Switch  equ     01h    ; ***'Switch' USED FOR A SWITCH ON LINE RB1 AND
                  00014                        ; *** AS A FLAG IN FLAGS2 FOR DEBOUNCING
                  00015
00000038          00016 SHIFT   equ     38h
00000012          00017 BANG    equ     12h
                  00018
00000008          00019 DEBOUNC equ     08h  ; *** #OF TIMES TO LOOP TO ALLOW DEBOUNCE OF SWITCH
                  00020
                  00021
                  00022
                  00023
                  00024 ; *** BIT ASSIGNMENTS FOR I/O LINES & TRI-STATING
                  00025
00000000          00026 ADB     equ     00h   ; Line used for ADB - pin XX (16C54)
00000001          00027 RA1     equ     01h   ; May be used as a Clock line TO another PIC
00000002          00028 RA2     equ     02h   ; May be used as a Clock line FROM another PIC
00000003          00029 RA3     equ     03h   ; May be used as a Data line between two PICs
                  00030
00000001          00031 TRI_IN  equ     01h   ; tri-state for ADB pin as input
00000000          00032 TRI_OUT equ     00h   ; tri-state for ADB pin as output
                  00033
                  00034
                  00035 ;*** MISC. CONSTANTS
                  00036
00000008          00037 BYTE    equ     08h   ; Receive 8 bits in Command; count from 8 to 0
00000002          00038 DEF_ADD equ     02h   ; default device address to start with (kybd)
00000003          00039 DEF_HND equ     03h   ; default Handler Id. to start with (std. kybd)
00000008          00040 OFFSET  equ     08h   ; offset to RAM address of the array of ADB
                  00041                       ; Data storage registers
                  00042
                  00043
                  00044 ;*** COMMAND MASKS:         MASK BITS FROM COMMAND REGISTER FOR:
                  00045
0000000F          00046 DEVMASK equ   0Fh     ;lower nibble holds Command (Talk, etc.) & Reg. #
000000F0          00047 ADDRMSK equ   0F0h    ;upper nibble holds the Device Address Number
0000000F          00048 CMDNIBL equ   0Fh     ;Command nibble from the address
0000000C          00049 CMDTYPE equ   0Ch     ;Upper 2 Command bits indicate Talk, Listen, etc.
00000003          00050 REGMASK equ   03h     ;Data Register Number bits from Command Nibble
0000001F          00051 FSRMASK equ   1Fh     ;FSR bits from the Command Nibble for RAM Address
                  00052
                  00053
                  00054 ;*** DATA COMMAND MASKS:    MASK DATA REGISTER 3a FOR:
                  00055
0000000F          00056 LOW_NBL equ     0Fh   ; Lower nibble from the 1st Data byte
000000F0          00057 HI_NIBL equ    0F0h   ; Upper nibble from the 1st Data byte
                  00058
                  00059
                  00060 ;*** CONSTANTS FOR MASKING OUT COMMAND NIBBLES (C_ indicates Command)
                  00061
                  00062                       ;   used to XOR if this is a:
0000000C          00063 C_TALK  equ     0Ch   ;   Talk Command
00000008          00064 C_LISTN equ     08h   ;   Listen Command
00000000          00065 C_RESET equ     00h   ;   Reset Command
00000001          00066 C_FLUSH equ     01h   ;   Flush Command
00000004          00067 C_RES_1 equ     04h   ;   Reserved Command 1
00000002          00068 C_RES_2 equ     02h   ;   Reserved Command 2
```

```
00000003          00069 C_RES_3 equ     03h   ;   Reserved Command 3
                  00070
                  00071
                  00072 ;*** DATA HANDLER ID MASKS: MASK DATA REGISTER 3b FOR:
                  00073
000000FF          00074 SELFTST equ    0FFh   ; Self-Test mode
00000000          00075 LISTEN1 equ      0h   ; unconditional address change
000000FE          00076 LISTEN2 equ    0FEh   ; address change if no collision detected
000000FD          00077 DEV_ACT equ    0FDh   ; address change if device activator is depressed
                  00078
                  00079
                  00080 ;BITS USED IN THE UPPER NIBBLE OF REGISTER 3a FOR SPECIAL ADB STATUS BITS
                  00081
00000004          00082 Resrvd3 equ     04h   ; reserved (Always 0)
00000005          00083 Srq_Bit equ     05h   ; determines if Host will accept Service Requests
00000006          00084 ExpEvnt equ     06h   ; indicates an Exceptional Event should take place
00000007          00085 Always0 equ     07h   ; always set to 0
                  00086
                  00087
                  00088 ;ADB FLAG BITS IN THE “FLAGS1” REGISTER (F1 indicates 1st Flags register)
                  00089
00000000          00090 F1Attn  equ     00h   ; set to know if 2nd Task taking place during Attn
00000001          00091 F1Reg3  equ     01h   ; Register 3 is being addressed
00000002          00092 F1Talk  equ     02h   ; indicates to Tlt routine this is a Talk Command
00000003          00093 F1Stop  equ     03h   ; set to indicate the Data Stop Bit is being sent
00000004          00094 F1Lstn  equ     04h   ; indicates to Tlt routine this is a Listen Command
00000005          00095 F1Sent1 equ     05h   ; 1st byte of Data Register has been sent
00000006          00096 F1Rcvd1 equ     06h   ; 1st byte of Data Register has been received
00000007          00097 F1Cllsn equ     07h   ; set to indicate that a collision occurred
                  00098
                  00099
                  00100 ;*** FLAG BITS IN THE “FLAGS2” REGISTER (F2 indicates 2nd Flags register)
                  00101
00000000          00102 F2Srq   equ     00h  ; indicate that Srq should be issued
                  00103 ;               01h Switch, defined above for PORT_B, also used as a Flag
00000002          00104 F2DActv equ     02h  ; change address if Device Activator is Depressed
00000003          00105 F2STest equ     03h  ; set to indicate a device Self Test to be performed
00000004          00106 F2SFail equ     04h  ; set to indicate that the Device Self-Test Failed
00000005          00107 F2DRcvd equ     05h  ; set when data is received for 2nd Application Task
00000006          00108 F2DSend equ     06h  ; set to indicate to Talk that Data needs to be sent
00000007          00109 F2DMore equ     07h  ;set in 2nd Task to indicate Data remains to be sent
                  00110
                  00111
                  00112 ;*** TIMING DEFINITIONS
                  00113                      ; These values currently used for clock at 4Mhz:
00000004          00114 PrSclr1 equ     .4   ;   this is used when TMR0 is being prescaled
00000001          00115 PrSclr2 equ     .1   ;   this is used when TMR0 is not prescaled
                  00116
000000C2          00117 ATT_MIN equ .776/PrSclr1 ; Attn lower limit:800 - 3% tolerance=776 usecs
000000CE          00118 ATT_MAX equ .824/PrSclr1 ; Attn upper limit:800 + 3% tolerance=824 usecs
0000007D          00119 TSK2MIN equ .500/PrSclr1 ; time given to 2nd Task during Attn Signal
000000E1          00120 TSK2MAX equ .900/PrSclr1 ;time given to 2nd Task after Data Sent/Received
00000048          00121 SYNC    equ .72/PrSclr2 ;Sync with extra tolerance after Attn detect
00000032          00122 BIT_TST equ .50/PrSclr2  ; if time is < 50 = 1 bit, & > 50 = 0 bit
00000048          00123 MAX_BIT equ .72/PrSclr2  ; Maximum time line can be low for a bit
00000068          00124 BITCELL equ .104/PrSclr2 ; Maximum time for a bit cell = 104 usecs
0000008C          00125 TLT_MIN equ .140/PrSclr2 ; Stop to Start minimum time = 140 usecs
000000FA          00126 TLT_MAX equ .250/PrSclr2 ; Stop to Start maximum time = 260 usecs
000000B4          00127 TLT_MID equ .180/PrSclr2 ; Stop to Start median time  = 208 usecs
0000004A          00128 SRQ_MAX equ .296/PrSclr1 ; amount of time to hold for a Service ReQuest
                  00129
                  00130 ;NOTE: for Timer0 timing of sending bits, some extra time is allowed for
                  00131 ;instruction cycles between the end of the bit and the start of the next
                  00132 ; bit
00000016          00133 LOW1BIT equ .22/PrSclr2  ; low time for a 1 bit
00000032          00134 HI_1BIT equ .50/PrSclr2  ; hi time  for a 1 bit
```

```
  00000038              LOW0BIT equ  .56/PrSclr2  ; low time for a 0 bit
  00000014              HI_0BIT equ  .20/PrSclr2  ; hi time  for a 0 bit


                        ;*** ADB DATA REGISTERS - 2 BYES FOR EACH OF REGISTERS 0, 1, 2, and 3

0008                    ADB_REG ORG   08h     ; ORIGIN FOR ADB DATA REGISTERS
0008                    Reg0a   RES   01h     ; 8
0009                    Reg0b   RES   01h     ; 9
000A                    Reg1a   RES   01h     ; A
000B                    Reg1b   RES   01h     ; B
000C                    Reg2a   RES   01h     ; C
000D                    Reg2b   RES   01h     ; D
000E                    Reg3a   RES   01h     ; E
000F                    Reg3b   RES   01h     ; F


                        ;* VARIABLE REGISTERS FOR STORAGE, FLAGS, THE TIME VARIABLE,
                        ;  THE COUNTER, & RANDOM VALUES

0010                    STORAGE ORG   10h  ; ORIGIN FOR MISC. DATA VARIABLES
0010                    TmpReg1 RES   01h  ; 10 - temporary registers where Data is sent from &
0011                    TmpReg2 RES   01h  ; 11 -  received; NOTE: THESE 2 MUST BE IN THIS ORDER
0012                    RegNum  RES   01h  ; 12 - holds current ADB Data Reg.#-NOT a RAM address
0013                    RAMaddr RES   01h  ; 13 - holds RAM address of ADB Data Reg.#
0014                    Flags1  RES   01h  ; 14 - two Flags registers used by ADB & 2nd
0015                    Flags2  RES   01h  ; 15 -   Application Task
0016                    CmdByte RES   01h  ; 16 - holds the Command Byte
0017                    BitCntr RES   01h  ; 17 - counts down when sending or receiving bits
0018                    Random  RES   01h  ; 18 - stores Random Address sent in Talk routine
0019                    TimeVar RES   01h  ; 19 - used with TMR0 for all ADB timing
001A                    Tsk2Var RES   01h  ; 1A - used with TMR0 for timing during 2nd Task


                        ;*** REGISTERS STILL AVAILABLE

001B                    TmpCtr1 RES   01h  ; 1B
001C                    TmpFlg1 RES   01h  ; 1C
001D                    TmpFlg2 RES   01h  ; 1C
001E                    TmpFlg3 RES   01h  ; 1D
001F                    TmpFlg4 RES   01h  ; 1E


0000                    PROGRAM ORG     00h     ; origin for program
                            include “adb.sub”   ; ADB Sub-Routines - these must be included
                        ;***********************************************************************
                        ;***********************************************************************
                        ;   ******           THE FOLLOWING ARE SUB-ROUTINES           *******
                        ;   ******             CALLED BY THE MAIN PROGRAM             *******
                        ;***********************************************************************
                        ;***********************************************************************

                        ;*** SWITCH PRESCALER BETWEEN WDT AND Timer0 *** (PrScale, NoPrScl) ***
                        ;*** THIS PROCEDURE, DOCUMENTED IN SPEC. SHEET SECTION 9.1, IS INTENDED
                        ;*** TO PREVENT UNEXPECTED RESET CONDITION

                        ;*** PrScale ROUTINE CALLED AT END OF AttnSig AND Srq SIGNALS
0000 0004               PrScale clrwdt             ; Change prescaler from WDT to TMR0
0001 0C01                       movlw   b’00000001’ ; BINARY - set to prescale TMR0
0002 0002                       option             ; Clear 4th bit from right to select TMR0
0003 0061                       clrf    TMR0       ; last 3 bits set prescale value as 1:4
0004 0800                       retlw   NULL       ;  this gives a good ratio to monitor the
                                                   ;  timing for Reset and Attention signals and
                                                   ;  the 2nd Application Task

                        ;***NoPrScl ROUTINE CALLED AT BEGINNING OF SyncSig AND END OF Srq SIGNALS
```

```
0005 0061           NoPrScl clrf    TMR0        ; Change prescaler from TMR0 to WDT
0006 0C08                   movlw   b'00001000' ; Set 4th bit from right to select WDT
0007 0002                   option
0008 0004                   clrwdt
0009 0C08                   movlw   b'00001000'
000A 0002                   option
000B 0800                   retlw   NULL

                    ;**********************************************************************

                    ;*GET INCOMING BIT & INTERPRET WHETHER IT'S A '1' OR A '0' *** (Get_Bit)*
                    ;*** Get_Bit CALLED BY COMMAND AND LISTEN ROUTINES
                    ; Get the bit, find out whether it's less than or greater than 50 usecs,
                    ; if < than 50 usecs, it's a '1' bit
                    ; if > than 50 usecs, it's a '0' bit
                    ; if it's a '1' bit, set LSB in the reg. pointed to by the FSR (Command
                    ; Byte) if it's a '0' bit, do nothing to the LSB
                    ; then look for the end of the Bit Cell (104 usecs max.)
                    ; if the maximum Bit time of (72 usecs) or maximum Bit Cell time is
                    ; exceeded, abort to the Attn Signal

000C 0201           Get_Bit movf   TMR0,W ; Check the time, then check if the line went high:
000D 0099                   subwf  TimeVar,W ; See if more than BIT_TST usecs have passed
000E 0703                   btfss  STATUS,C  ; if not, check whether the line went high
000F 0AAB                   goto   AttnSig   ; if so, abort to the Attn Signal
0010 0705                   btfss  PORTA,ADB ; Check whether the line went high
0011 0A0C                   goto   Get_Bit   ; if line is still low, loop again
0012 0C32                   movlw  BIT_TST   ; if line went high, see if it's a '1' or a '0'
0013 0039                   movwf  TimeVar   ; as the bit has not yet been determined yet,
0014 0400                   bcf    INDF,LSB  ; ensure the LSB in the indirect address is '0'
0015 0201                   movf   TMR0,W    ; Get the time
0016 0099                   subwf  TimeVar,W ;  if time < 50 usecs, it's a '1' bit
0017 0603                   btfsc  STATUS,C  ; if time > 50 usecs and < 72, it's a '0' bit
0018 0500                   bsf    INDF,LSB  ; if it's a 1, set LSB in the address FSR points
0019 0C68                   movlw  BITCELL   ; to Check whether the Max. Bit Cell time of
001A 0039                   movwf  TimeVar   ; 104 usecs has been exceeded
001B 0201           CellChk movf   TMR0,W    ; Check the time, then check the line
001C 0099                   subwf  TimeVar,W ; See if more than Max. Bit Cell usecs have
001D 0703                   btfss  STATUS,C; passed if not, look for the line to go low again
001E 0AAB                   goto   AttnSig   ; if so, abort to the Attn Signal or Reset
001F 0605                   btfsc  PORTA,ADB ; Check the line for the start of another bit
0020 0A1B                   goto   CellChk   ; if the line is still high, loop CelChk1 again
0021 0061                   clrf   TMR0      ; if the line went low, clear the TMR0 & return
0022 0800                   retlw  NULL      ; for another bit or to interpret the Command

                    ;**********************************************************************
                    ;* DETERMINE IF THIS IS A GLOBAL COMMAND TO ALL DEVICES  *** (Globals) *
                    ;*** Globals CALLED BY AddrChK

0023 0211           Globals movf   TmpReg2,W ; Check whether the Command is for all devices
0024 0F04                   xorlw  C_RES_1   ; retrieve the Command Type (the upper 2 bits
0025 0643                   btfsc  STATUS,Z  ; of the Command nibble)
0026 0BC4                   goto   Reserv1   ; test for this being the first Reserved
0027 0210                   movf   TmpReg1,W ; Command retrieve the whole Command Nibble
0028 0F02                   xorlw  C_RES_2 ; test for this being the second Reserved Command
0029 0643                   btfsc  STATUS,Z
002A 0BC5                   goto   Reserv2
002B 0210                   movf   TmpReg1,W ; retrieve the whole Command Nibble
002C 0F03                   xorlw  C_RES_3   ; test for this being the third Reserved Command
002D 0643                   btfsc  STATUS,Z
002E 0BC6                   goto   Reserv3
002F 0F00                   xorlw  C_RESET   ; test for this being Reset Command
0030 0643                   btfsc  STATUS,Z
0031 0A96                   goto   Reset
0032 0800                   retlw  NULL

```

```
                 ;***********************************************************************
                 
                 ;* MASK OUT COMMAND NIBBLE AND REG.# BITS FROM THE COMMAND *** (MaskCmd)*
                 ; NOTE: This routine should only be called once during any single ADB
                 ; transaction, from either AddrChk or CmmdChk
                 
0033 0216        MaskCmd movf  CmdByte,W ; Mask the Command to save the Data Reg. # bits &
0034 0E0F                andlw CMDNIBL   ; the Command Type bits (Listen, Talk, etc.):
0035 0030                movwf TmpReg1   ; save the Command nibble
0036 0E0C                andlw CMDTYPE   ; mask the upper 2 Command Type bits (Talk, etc.)
0037 0031                movwf TmpReg2   ; save the upper 2 Command Type bits
0038 0216                movf  CmdByte,W : extract the Data Register number:
0039 0E03                andlw REGMASK   ; mask out Data Reg. number from Command Nibble
003A 0032                movwf RegNum    ; save the Data Reg. bits
003B 0024                movwf SR        ; save pointer to Data Reg. in File Select Reg.
                                         ; in order to setup RAM address where start
                                         ; of Data for this Reg. will be stored
003C 0403        SaveRAM bcf     STATUS,C; clear Carry bit so it doesn't wrap around
003D 0364                rlf     FSR,F   ; multiply by 2 to get 1st Byte of RAM addr
003E 0564                bsf     FSR,03h ; add array offset for Send/Receive/Flush Reg.
003F 0204                movf    FSR,W   ; by setting bit of 1st RAM address, which
0040 0E1F                andlw   FSRMASK ; is ORG'd in ADB.EQU equates
0041 0033                movwf   RAMaddr ; mask out the RAM address of Data Reg. Number
0042 0800                retlw   NULL    ; save RAM address of Data Reg. and return
                 
                 ;************************************************************************
                 
                 ;*** ISSUE A SERVICE REQUEST IF NECESSARY *** (Srq; may call LineLow) ***
                 ;*** CALLED BY AddrChk
                 ; see if the Srq Flag is set, if not, return, otherwise:
                 ; change the prescaler to TMR0 since this takes longer than 255 usecs,
                 ; load the SRQTIME of 300 usecs into the TimeVariable,
                 ; call LineLow to:
                 ; keep checking the time to see if 300 usecs have passed,
                 ; let the line go high again,
                 ; and see if the line is high, and if not, abort, if it is,
                 ; change the prescaler back to WDT, and return
                 
0043 0715        Srq     btfss   Flags2,F2Srq ; see if the Srq flag is set,
0044 0800                retlw   NULL         ; if not, return
0045 0900                call    PrScale      ; switch the prescaler to TMR0
0046 0C00                movlw   TRI_OUT      ; tri-state PORTA to make the ADB an output
0047 0005                tris    PORTA
0048 0C4A                movlw   SRQ_MAX
0049 0976                call    LineLow
004A 0905                call    NoPrScl      ; change the prescaler back to WDT
004B 0800                retlw   NULL
                 
                 ;************************************************************************
                 
                 ;*** Tlt - TIME FROM STOP BIT TO START BIT ***  (Tlt) ***
                 ;*** CALLED BY EITHER Talk OR Listen ROUTINES
                 ; Loop checking the time, then checking the line to see if it went low
                 ; if at any time the line goes low,
                 ; see if this is a Talk Command,
                 ; if it is a Talk Commmand, go to the Collision routine
                 ; if the line goes low before the minimum Tlt time, abort to Attn Signal
                 ; if the line is high longer than TLT_Min usecs,
                 ; see if this is a Talk Command, and if it is, wait for the mid-point,
                 ; and return to Send the Start Bit, Data Bytes, & the Stop Bit
                 ; if it's not a Talk Command, see if it's a Listen Command, and if so,
                 ; load Tlt_Max for TimeVariable, and look for the line to go
                 ; low as the beginning of the Start Bit,
                 ; if more than Tlt_Max usecs pass, abort to Attn Signal
                 ; if the line goes low and this is a Listen Command,
                 ; clear the TMR0 & return to get the rest of the Start Bit
```

```

004C 0C8C         Tlt     movlw   TLT_MIN      ; Look for Stop-to-Start-Time, Tlt
004D 0039                 movwf   TimeVar      ; Check the time, then check the line
004E 0201         TltChk1 movf    TMR0,W       ; See if more than TLT_MIN usecs have passed
004F 01B8                 xorwf   Random,F     ; (ensure the Talk R3 address is Random with
0050 0099                 subwf   TimeVar,W    ; XOR) by checking whether Carry bit is set
0051 0703                 btfss   STATUS,C     ; after subtraction
0052 0A5D                 goto    ChkFlag      ; if TLT_MIN usecs passed, see what Command
0053 0605                 btfsc   PORTA,ADB    ; this is if not, check whether the line went
0054 0A4E                 goto    TltChk1      ; low if the line is still high, keep looping
0055 0654                 btfsc   Flags1,F1Talk;  if line went low, see if this is a Talk
0056 0B5A                 goto    Collisn      ; Command if it is, there was a Collision,
0057 0201                 movf    TMR0,W       ; abort otherwise, check the time
0058 0099                 subwf   TimeVar,W    ; see if TLT_MIN usecs passed,
0059 0703                 btfss   STATUS,C     ; if not, abort to Attn Signal, too little
005A 0AAB                 goto    AttnSig      ; time passed when the line went low
005B 0061                 clrf    TMR0         ; if it’s not a Talk Command, clear the TMR0
005C 0800                 retlw   NULL         ; and return for the rest of the Start Bit

005D 0654         ChkFlag btfsc   Flags1,F1Talk ; Check whether to Talk or Listen
005E 0A6D                 goto    TltTalk       ; if Talk, wait for mid-point of Tlt time
005F 0794                 btfss   Flags1,F1Lstn ; if Listen, continue to look for Start Bit
0060 0800                 retlw   NULL          ; if neither flag is set, abort, something’s
0061 0CFA                 movlw   TLT_MAX       ; wrong  Load TimeVariable to check for
0062 0039                 movwf   TimeVar       ; upper limit of Tlt time
0063 0201         TltChk2 movf    TMR0,W        ; See if TLT_MAX usecs have been exceeded
0064 0099                 subwf   TimeVar,W     ; by checking whether Carry bit is set
0065 0703                 btfss   STATUS,C      ; after subtraction
0066 0AAB                 goto    AttnSig       ; if so, abort to Attn Signal
0067 0605                 btfsc   PORTA,ADB     ; if not, check whether the line went low
0068 0A63                 goto    TltChk2       ;if line is still high, check the time again
0069 0654                 btfsc   Flags1,F1Talk ;if line went low, see if this is a Talk
006A 0B5A                 goto    Collisn       ; Command if so, there was a Collision
006B 0061                 clrf    TMR0          ; if it’s not a Talk Command, return to get
006C 0800                 retlw   NULL          ; the rest of the Start Bit from Host

006D 0CB4         TltTalk movlw   TLT_MID       ; Load TimeVariable so Talk will send Start
006E 0039                 movwf   TimeVar       ; Bit at about the mid-point of the Tlt
006F 0201         TltChk3 movf    TMR0,W        ; See if TLT_MID usecs have been exceeded
0070 0099                 subwf   TimeVar,W     ; by checking whether Carry bit is set
0071 0703                 btfss   STATUS,C      ; after subtraction
0072 0800                 retlw   NULL        ; if time was exceeded, return to send Start Bit
0073 0605                 btfsc   PORTA,ADB     ; if not, check whether the line went low
0074 0A6F                 goto    TltChk3       ; if line is still high, check the time again
0075 0B5A                 goto    Collisn       ; if the line went low, abort to Collision

                  ;***********************************************************************

                  ;*** MAKE LINE GO LOW TIME IN TimeVar AS A ‘1’ OR ‘0’ BIT***(LineLow)***
                  ;*** CALLED BY Talk OR Srq

0076 0039         LineLow movwf   TimeVar     ;
0077 0201         Low_Tmp movf    TMR0,W      ; Check the clock,
0078 0099                 subwf   TimeVar,W   ; loop until TimeVar usecs have passed
0079 0603                 btfsc   STATUS,C;
007A 0A77                 goto    Low_Tmp     ;
007B 0C01                 movlw   TRI_IN      ; Tri-state PORTA to make ADB line an input
007C 0005                 tris    PORTA       ; again and let the line go high
007D 0061                 clrf    TMR0        ; and clear TMR0
007E 0000                 nop                 ; Allow the ADB Port line to stabilize
007F 0000                 nop                 ; Allow the ADB Port line to stabilize
0080 0705                 btfss   PORTA,ADB   ; check if the line is still low, if so, a
0081 0B5A                 goto    Collisn     ; Collision occurred
0082 0800                 retlw   NULL   ; if not, return to load high time for rest of bit

                  ;* MAKE LINE GO HIGH FOR REST OF BIT CELL TIME IN TimeVar *** (LineHi)*
```

```
              ;*** CALLED BY Talk

0083 0039     LineHi  movwf   TimeVar  ; Let the line go high for a pre-designated time
0084 0201     Hi_Tmp  movf    TMR0,W   ; Check the clock,
0085 0099             subwf   TimeVar,W; loop until TimeVar usecs have passed
0086 0603             btfsc   STATUS,C ;
0087 0A84             goto    Hi_Tmp   ;
0088 0705             btfss   PORTA,ADB; check if the line is still high,
0089 0B5A             goto    Collisn  ; if not, a Collision occurred, Abort
008A 0674             btfsc   Flags1,F1Stop; if this is the end of the Data Stop Bit,
008B 0800             retlw   NULL     ; don’t let the line go low again, just return
008C 0C00             movlw   TRI_OUT  ; if still high, start sending a bit to the Host
008D 0005             tris    PORTA    ; tri-state PORTA to make the ADB an output and
008E 0061             clrf    TMR0     ; return
008F 0800             retlw   NULL     ;

              ;************************************************************************
              ;************************************************************************
              ;   ******                  END OF SUB-ROUTINES                *******
              ;************************************************************************
              ;************************************************************************
                                       ; here to ensure being in the first
                                       ; half of the memory page when called.

              IntData macro DataCmd,Routine ; Macro goes to an appropriate Listen Reg.3
                      movf    TmpReg2,W ; interprets the Data Command received by
                      xorlw   DataCmd   ; comparing the 2nd byte to a Data
                      btfsc   STATUS,Z  ; Command constant
                      goto    Routine   ; it then goes to the appropriate routine
                      endm

              ;
              ;*** CONDITIONAL ASSEMBLY DETERMINED BY LIST DIRECTIVE
              ;
                      ifdef      __16C56
                      include “5657mcro.mod”     ; macros for the 2nd Application Task
              ;
              ;*** LoadEm MACRO USED FOR TESTING DURING 2ND APPLICATION TASK
              ;*** ONLY FOR PART 16c56/57
              ;
  00000004    H       equ     04h     ; *** THESE ARE USED AS KEYS PRESSED WHEN PART
  0000000E    E       equ     0Eh     ; ***   IS SELECTED FOR 16C56/57
  00000025    L       equ     25h
  0000001F    O       equ     1Fh
  00000031    SP      equ     31h
  0000000D    WW      equ     0Dh     ; W is already defined in the PICREG5X.EQU file
  0000000F    R       equ     0Fh
  00000002    D       equ     02h
  00000024    RETRN   equ     24h
  000000FF    FILLCHR equ     0FFh    ; ‘fill character’ as described in spec.
              ;
              ;
              LoadEm  macro   Ctr,Bit,Dest,RegA,RegB; Macro used to load registers and
                      btfss   Ctr,Bit       ;  set  flags for Key-Up Transition Codes
                      goto    Dest          ; Bits are cleared as the data is sent
                      movlw   Reg0a
                      movwf   FSR
                      movlw   RegA          ; load data to be sent from register A
                      movwf   INADDR
                      incf    FSR,F
                      movlw   RegB          ; load data to be sent from register B
                      movwf   INADDR        ; load data to be sent from register B
                      bsf     Flags2,F2DSend; Data now needs to be sent to the host
                      bsf     Flags2,F2Srq  ; Until all data has been sent, Srq’s may
                      btfsc   Flags2,F2STest; be sent. See if Key Transition Codes
                      goto    KeyUp         ; should be sent if so, go set the bits
```

```
                  bsf     Flags2,F2STest; if not, set bits so they’ll be next time
                  bcf     Ctr,Bit       ; clear the bit so next data will be sent
                  goto    DBounce       ; and go debounce the switch
                  endm



                  endif                  ;   the program is for a 16C56
                  ifdef      __16C57     ;   or 16C57 part
                  include “5657mcro.mod”
                  endif

            ;*******************************************************************
            ;*******************************************************************
            ;               THE MAIN PROGRAM STARTS BELOW
            ;*******************************************************************
            ;*******************************************************************

0090 0C01   Start movlw   TRI_IN    ; Start off by making the ADB pin an
0091 0005         tris    PORTA     ; input on PORTA
0092 0405         bcf     PORTA,ADB ; make line will go low when tris’d as an output

            ; *** THIS I/O SETUP ROUTINE IS USED FOR TESTING WITH AN LED ON RB0
            ; *** AND A SWITCH ON RB1
0093 0C02   TSTING1 movlw   b’00000010’; Make RB0 an output (for the LED) and
0094 0006         tris    PORTB     ; RB1 an input (for the normally open switch)
0095 0406         bcf     PORTB,LED ; Make sure the LED is off to begin with

            ;*******************************************************************

0096 0705   Reset btfss   PORTA,ADB ; Reset Signal - loop until the line is high,
0097 0A96         goto    Reset     ; then initialize Registers

0098 0070   Init  clrf    TmpReg1   ; Initialization routine
0099 0071         clrf    TmpReg2   ; Clear variables
009A 0072         clrf    RegNum    ; NOTE: No need to clear variable register
009B 0073         clrf    RAMaddr   ;  ‘Random’ as it is XOR’d in other routines
009C 0074         clrf    Flags1    ;  to produce a random Address for the ‘Talk
009D 0075         clrf    Flags2    ;  Reg. 3’ Command
009E 0077         clrf    BitCntr
009F 0068         clrf    Reg0a     ; Clear ADB Storage Data Register Variables
00A0 0069         clrf    Reg0b
00A1 006A         clrf    Reg1a
00A2 006B         clrf    Reg1b
00A3 006C         clrf    Reg2a
00A4 006D         clrf    Reg2b

00A5 0C02         movlw   DEF_ADD   ; Register 3 has special Default Data set at
00A6 002E         movwf   Reg3a     ; Reset: load Register 3a with Default Device
00A7 05AE         bsf     Reg3a,Srq_Bit; Address allow Service Requests of Host
00A8 05CE         bsf     Reg3a,ExpEvnt;  include the Exceptional Event bit as
                                    ;default * NOTE: at this time, this Device
                                    ; doesn’t process for Exceptional Events
00A9 0C03         movlw   DEF_HND   ;
00AA 002F         movwf   Reg3b     ; load Register 3b with Default Device Handler ID

            ;*******************************************************************

            ;*** LOOK FOR ATTENTION OR RESET ***  (AttnSig)  ***
            ; Look for the line being low, when it is, see if the line went high.
            ; During that time, allow the 2nd Application Task to be performed for a
            ; limited amount of time, then return to Attn Signal
            ; if the line went high, did it go high within the 776-824 usec range?
            ; if so, go on to get the Command
            ; if not, goto the Reset routine
            ; IN DETAIL:
              ;   look at the line
              ;   if the line is not yet low,
              ;   loop until it goes low, & clear the TMR0
              ;   Loop with Minimum Time: check the time
              ;   if the time is less than the Attention Minimum usecs,
              ;   check whether the line has gone high,
              ;   if the line has not gone high,
              ;   loop again checking the time
              ;   if the line has gone high,
              ;   check whether the Min. usecs have passed
              ;   if not, Abort; too little time went by.
              ;   if so, go on to look for the Sync signal
              ;   Loop with Maximum Time: load the Maximum Time Variable & check
              ;   the time if the time is less than the Attention Maximum usecs,
              ;   check whether the line has gone high,
              ;   if the line has not gone high,
              ;   loop again checking the time
              ;   if the line has gone high before Max. Attention usecs have passed,
              ;   go on to look for the Sync signal
              ;   if the time is greater than the Attention Maximum usecs,
              ;   abort to Reset

              ;**********************************************************************

00AB 0201     AttnSig movf   TMR0,W        ; Look for Attn between ATT_MIN - ATT_MAX usecs
00AC 07F4             btfss  Flags1,F1Cllsn; this is a good time to use the TMR0 and
00AD 01B8             xorwf  Random,F      ; Pseudo-Random Address
00AE 0605             btfsc  PORTA,ADB     ; See if the line went low
00AF 0AAB             goto   AttnSig       ; Loop to AttnSig until the line goes low
00B0 0900             call   PrScale       ; Switch prescaler to TMR0 for > 250 usec count
                                           ; during Attn Signal
00B1 0CC2             movlw   ATT_MIN      ;
00B2 0039             movwf   TimeVar      ; use TimeVariable to subtract from ATT_MIN usecs

00B3 0076     CleanUp clrf    CmdByte      ; Clear the Command Byte
00B4 0070             clrf    TmpReg1      ; Clear the temporary Data registers
00B5 0071             clrf    TmpReg2      ; NOTE: No need to clear variable register
00B6 0072             clrf    RegNum       ; ‘Random’ clear the current Register Number
00B7 0073             clrf    RAMaddr      ; register clear the register holding the RAM
                                           ; Address of the 1st byte of where Data is stored
00B8 0514             bsf     Flags1,F1Attn; Set this bit to indicate to the 2nd Task
                                           ; that it should Return to the AttnMin routine
00B9 0434             bcf     Flags1,F1Reg3    ; Clear Flags: Data-for-Register 3
00BA 0454             bcf     Flags1,F1Talk    ; Talk
00BB 0474             bcf     Flags1,F1Stop    ; Data-Stop-Bit-is-being-sent
00BC 0494             bcf     Flags1,F1Lstn    ; Listen
00BD 04B4             bcf     Flags1,F1Sent1   ; Sent-1st-Byte
00BE 04D4             bcf     Flags1,F1Rcvd1   ; Received-1st-Byte

00BF 0C7D             movlw   TSK2MIN      ; load Task 2 Time Variable with amount allowed
00C0 003A             movwf   Tsk2Var      ; during Attn Signal
00C1 0BCB             goto    Task_2       ; This space allows running a second application
                                           ; NOTE: BE SURE TO RETURN TO ATTNMIN BEFORE 750
                                           ; usecs HAVE PASSED, AND DON’T LET THE OTHER
                                           ; APPLICATION AFFECT THE Timer0 or TimeVar.

00C2 0201     AttnMin movf    TMR0,W       ; Check the time, then check the line
00C3 0099             subwf   TimeVar,W    ; See if more than ATT_MIN usecs have passed
00C4 0703             btfss   STATUS,C     ; if not, check the line
00C5 0ACD             goto    AttnMax      ; if so, go check time/line again in AttnMax
00C6 0705             btfss   PORTA,ADB    ; Check for line being high & if so, check time
00C7 0AC2             goto    AttnMin      ; if line is still low, loop again
00C8 0201             movf    TMR0,W       ; if line is high, see if time is in range
00C9 0099             subwf   TimeVar,W    ; by checking whether Carry bit is
00CA 0703             btfss   STATUS,C     ; set after subtraction
00CB 0AAB             goto    AttnSig      ; If time <= Min, look for Attn Signal again
```

```
00CC 0AD6                 goto    SyncSig   ; If time > Min, go get Sync signal

00CD 0CCE         AttnMax movlw   ATT_MAX   ; Load the TimeVariable to check for the
00CE 0039                 movwf   TimeVar   ; maximum amount of time for Attn Signal
00CF 0201         AttnTmp movf    TMR0,W    ; Check the time, then check the line
00D0 0099                 subwf   TimeVar,W ; See if more than ATT_MAX usecs have passed
00D1 0703                 btfss   STATUS,C  ; if not, check the line
00D2 0A96                 goto    Reset     ; if so, Abort to Reset;too much time has passed
00D3 0705                 btfss   PORTA,ADB ; Check for the line to going high
00D4 0ACF                 goto    AttnTmp   ; if the line isn't high, loop AttnMax again
00D5 0061                 clrf    TMR0      ; if the went high, go get the Sync signal

                  ;*********************************************************************

                  ;*** LOOK FOR SYNC SIGNAL ***  (SyncSig) ***
                  ; This routine checks the timing between the rising edge of the Attention
                  ; Signal & a falling edge indicating the start of the 1st Command bit.
                  ; At the end of the Attn Signal routine, the line went high, and
                  ; the TMR0 was cleared.
                  ; Check the TMR0,
                  ; if the 72 usec limit is exceeded,
                  ; abort to the Attn Signal
                  ; if the 72 usec limit is not exceed,
                  ; check the line
                  ; if the line went low (as the first bit of the Command),
                  ; go on to get the 8 Command Bits
                  ; if the line is still high,
                  ; loop to check TMR0 again

                  ;*********************************************************************

00D6 0905         SyncSig call    NoPrScl   ; Get the Sync Signal which follows the Attn
00D7 0C48                 movlw   SYNC      ; Signal Turn off prescaler; timing counts are
00D8 0039                 movwf   TimeVar   ; < 255 usecs  and load the timing the for the
00D9 0099         SyncTmp subwf   TimeVar,W ; Sync Signal See if more than SYNC usecs
00DA 0703                 btfss   STATUS,C  ; have passed if not, go check the line
00DB 0AAB                 goto    AttnSig   ; if so, Abort to Attn Signal
00DC 0605                 btfsc   PORTA,ADB ; Check for the line to go low
00DD 0AD9                 goto    SyncTmp   ; if the line is still high, loop again
00DE 0061                 clrf    TMR0      ; if low, clear TMR0 & go on to get the Command

                  ;*********************************************************************

                  ;*** GET THE COMMAND: 8 BITS & STOP BIT ***  (Command) ***
                  ; The Sync Signal was detected when the line went low after approximately
                  ; 70 usecs.  This low line is the first bit of the Command.  This
                  ; routine receives 8 bits, followed by a '1' Stop bit.

                  ; IN DETAIL:
                  ; initialize a counter for counting down as the bits come in
                  ; call Get_Bit to receive each bit, MSB first, & rotate it into the
                  ; CmdByte register, where the Command Byte is stored.
                  ; After returning from GetBit, decrement the counter.
                  ; when all 8 bits have been received, clear TMR0 (to allow looking
                  ; for the Stop bit, or holding down the line for an SRQ), and go on to
                  ; Interpret the Command.

                  ; In GetBit, get the time,
                  ; if the time is greater than 72 usecs,
                  ; abort to the Attn Signal
                  ; if the time is less than 72 usecs,
                  ; check if the line went high
                  ; if line is still low,
                  ; loop to check the time again
                  ; if the line went high,
                  ; determine whether the line went high before or after 50 usecs
```

```
                ; if the line went high before 50 usecs, rotate a 1 bit into CmdByte reg.
                ; if the line went high after 50 usecs, rotate a 0 bit into CmdByte reg.
                ; after getting a bit, check if the line went low (the start of the next
                ; bit) if the max. Cell Bit time (104 usecs) is exceeded, abort to Attn
                ; Signal when the line goes low, clear TMR0 and return to get another
                ;  bit or  interpret the Command if all 8 bits have been been received

                ;***********************************************************************

00DF 0C08       Command movlw   BYTE      ; Get the 8 Command Bits - 1st bit already
00E0 0037               movwf   BitCntr   ; started, so count down from 8 to 0
00E1 0C16               movlw   CmdByte   ; rotate bits into CmdByte with indirect
00E2 0024               movwf   FSR       ; address
00E3 0C48       CmdLoop movlw   MAX_BIT   ; Get & rotate a 1 or 0 bit into CmdByte, or
00E4 0039               movwf   TimeVar   ; see if the maximum time is exceeded & abort
00E5 0403               bcf     STATUS,C  ; clear Carry bit to ensure it won’t wrap around
00E6 0376               rlf     CmdByte,F ; rotate in the last bit
00E7 090C               call    Get_Bit   ; and get another one
00E8 02F7               decfsz  BitCntr,F ; keep looping until 8 bits are received &
00E9 0AE3               goto    CmdLoop   ; rotated when the Command has been received,
                                          ; interpret it
                ;***********************************************************************

                ;*** CHECK THE ADDRESS ***  (AddrChk; may call MaskCmd, Globals, Srq) ***
                ; The Command Stop Bit is a good time to determine if the Host is
                ; addressing this Device:
                ; test the left nibble of the received byte against the current Address
                ; if the Address belongs to this Device,
                ; mask out the command and register nibble of the received byte,
                ; test it to see whether the Command is to Listen, Talk, or Flush
                ; and go to the routine that looks for the end of the Stop Bit
                ; if the Command is for another Device,
                ; mask the command nibble
                ; see if the Command is a global/reserved Command
                ; if so, go do the Command
                ; if the Command is not global,
                ; check the Srq flag to see if another application needs service
                ; if the Srq flag is set,
                ; go issue a Service Request (Srq)
                ; if the Srq flag is not set,
                ; go get the Attn Signal

00EA 020E       AddrChk movf   Reg3a,W   ; See if the Command received is for this Device
00EB 0E0F               andlw  DEVMASK   ; by masking off this Device’s Address
00EC 0031               movwf  TmpReg2   ; and saving it in a temporary register
00ED 03B1               swapf  TmpReg2,F ; (received nibbles in Command are reversed)
00EE 0216               movf   CmdByte,W ; Test if the received Address is for Device,
00EF 0EF0               andlw  ADDRMSK   ; by masking out the Command nibble,
00F0 0191               xorwf  TmpReg2,W ; compare received Address to current Address
00F1 0643               btfsc  STATUS,Z  ; if Address is for this Device, go get the Stop
00F2 0AF7               goto   CmdStop   ; Bit & see what the Command is for this Device.

00F3 0933               call   MaskCmd   ; Mask the Command Nibbles from the Address
00F4 0923               call   Globals   ; and go see if it was a Global Command
00F5 0943               call   Srq       ; if not, go see if Srq needs to be asserted
00F6 0AAB               goto   AttnSig   ; if not, go get the Attn Signal

                ;**********************************************************************

                ;*** LOOK FOR THE COMMAND STOP BIT *** (CmdStop) ***
                ; Look for the Stop Bit following the Command Byte.  This is not executed
                ; if Srq is asserted by this Device.

00F7 0C48       CmdStop movlw   MAX_BIT   ; load the maximum time for a bit low time
00F8 0039               movwf   TimeVar   ;
00F9 0099               subwf   TimeVar,W ; See if more than the max. # of usecs have
```

```
00FA 0703               btfss   STATUS,C    ; passed if not, go check for the line to go
00FB 0AAB               goto    AttnSig     ; high if so, abort to the Attn Signal
00FC 0705               btfss   PORTA,ADB   ; Check for the line to go high
00FD 0AF7               goto    CmdStop     ; if the line is still low, loop CmdStop
00FE 0061               clrf    TMR0        ; again if high, clear TMR0 as the beginning
                                            ; of the Tlt and go on to interpret Command
                                            ; as Talk, Listen, or Flush.

                ;************************************************************************

                ;*** INTERPRET THE COMMAND ***  (CmmdChk) ***
                ; Determine first if the command is for Register 3, and set the Reg3 flag
                ; if so, then see if the Command is to Talk, Listen, or Flush and go to
                ; that routine.

00FF 0933       CmmdChk call    MaskCmd     ; Separate the Command Nibbles into temp. regs.
0100 0712               btfss   RegNum,00h  ; (MaskCmd put Command Type bits into TmpReg1)
0101 0B04               goto    CmdChk2     ; see if the Command is for Register 3
0102 0632               btfsc   RegNum,01h  ; if not, go continue interpreting the Command
0103 0534               bsf     Flags1,F1Reg3; if so, set the Reg. 3 flag indicating this
                                            ; condition for the Talk or Listen routines

0104 0211       CmdChk2 movf    TmpReg2,W   ; Test what Command was received &
0105 0F0C               xorlw   C_TALK      ;   branch accordingly
0106 0643               btfsc   STATUS,Z    ; test for this being a Talk Command
0107 0B11               goto    Talk
0108 0211               movf    TmpReg2,W
0109 0F08               xorlw   C_LISTN
010A 0643               btfsc   STATUS,Z    ; test for this being a Listen Command
010B 0B5D               goto    Listen
010C 0211               movf    TmpReg2,W
010D 0F01               xorlw   C_FLUSH
010E 0643               btfsc   STATUS,Z    ; test if the Command is to Flush a Register
010F 0BBE               goto    Flush       ; if the Command isn’t a Flush, go get
0110 0AAB               goto    AttnSig     ;   the Attn Signal

                ;************************************************************************

                ;*** SEND DATA TO THE HOST *** (Talk; calls Tlt, LineLow, LineHi) ***
                ; Data is sent to Host from ADB Data Registers using indirect addressing.
                ; (TMR0 was cleared in CmmdChk, and timing for Tlt began there)
                ; Call the Tlt (Stop to Start Time), which waits for the middle of the
                ; Tlt, when the Tlt returns, send a ‘1’ Start Bit,
                ; load the first byte of the Data Register into temporary register,
                ; send the 1st 8 bits,
                ; load the second byte of the Data Register into temporary register,
                ; send the 2nd 8 bits,
                ; and send a ‘0’ Stop Bit
                ; if at anytime during the Tlt, LineLow, or LineHi the ADB line is
                ; inappropriately high or low, the routine aborts to the Collision
                ; routine. The Collision routine only sets a flag if this is a Talk Reg.
                ; 3 Command, indicating a Collision occurred when sending Data for Reg.
                ; 3, and goes  to get the Attention Signal.
                ; Using temporary registers ensures the Data doesn’t get cleared until
                ; all of it has been sent.

0111 0634       Talk  btfsc  Flags1,F1Reg3  ; if the talk command is for Register 3,
0112 0B1D             goto   SetRndm        ; go create  a Random Address and load it into
0113 07D5             btfss  Flags2,F2DSend ; TmpReg1 Check whether there is data to
0114 0AAB             goto   AttnSig        ; send if not, let the bus timeout & get Attn

0115 0213       SetTmps movf  RAMaddr,W     ; Signal Load the temporary registers with Data
0116 0024               movwf FSR           ; stored at the appropriate RAM Address for the
0117 0200               movf  INDF,W        ; Register indicated in the Command Byte
0118 0030               movwf TmpReg1
0119 02A4               incf  FSR,F
```

```
011A 0200     00587         movf  INDF,W          ;Load 2nd temporary register from 2nd RAM
011B 0031     00588         movwf TmpReg2         ;Address where Data is stored
011C 0B29     00589         goto  CallTlt
              00590
011D 0201     00591 SetRndm movf  TMR0,W          ;The Address sent to the Host for a Talk Reg.3
011E 0198     00592         xorwf Random,W        ;Command must be random to avoid collisions
011F 0E0F     00593         andlw LOW_NBL         ;with other Device Addresses during
0120 0030     00594         movwf TmpReg1         ;initialization
0121 020E     00595         movf  Reg3a,W         ;
0122 0EF0     00596         andlw HI_NIBL         ;
0123 0130     00597         iorwf TmpReg1,F       ;
0124 0634     00598 SetHndl btfsc Flags1,F1Reg3   ; if this is a Talk R3 Command,
0125 020F     00599         movf  Reg3b,W         ; send the Device Handler ID
0126 0695     00600        btfsc  Flags2,F2SFail  ; if a Device Self-Test was performed and it
0127 0040     00601         clrw                  ; failed, send the reserved Handler ID of
0128 0031     00602         movwf TmpReg2         ; ‘00h’ to indicate the Failed condition
              00603
0129 0554     00604 CallTlt bsf   Flags1,F1Talk   ; Set the Talk Flag to indicate to the Tlt
012A 094C     00605         call  Tlt             ; routine to return for the end of Talk Start Bit
              00606
012B 0C10     00607 SndStrt movlw TmpReg1         ; Send a ‘1’ bit as the Start Bit
012C 0024     00608         movwf FSR             ; Use the indirect addressing of the temporary
012D 0C00     00609         movlw TRI_OUT         ; registers from which Data will be sent
012E 0005     00610         tris  PORTA           ; tri-state PORTA to make the ADB an output
012F 0061     00611         clrf  TMR0            ; clear TMR0 as the beginning of a bit
0130 0C16     00612         movlw LOW1BIT
0131 0976     00613         call  LineLow         ; hold the line low for 1/3rd of a Bit Cell
0132 0C32     00614         movlw HI_1BIT
0133 0983     00615         call  LineHi          ; let the go line high for rest of the Bit Cell
              00616
0134 0C08     00617 SetSend movlw BYTE            ; Send the data bytes
0135 0037     00618         movwf BitCntr         ; Load the counter to send 8 Bits
0136 06E0     00619 SndBits btfsc INDF,MSB        ; determine whether to complete the send of
0137 0B3D     00620         goto  Send1           ; a ‘1’ or ‘0’ bit
              00621
0138 0C38     00622 Send0   movlw LOW0BIT         ; Send a ‘0’ bit
0139 0976     00623         call  LineLow         ; hold the line low for 2/3rd of a Bit Cell
013A 0C14     00624         movlw HI_0BIT
013B 0983     00625         call  LineHi          ; let the line high for the rest of the Bit Cell
013C 0B41     00626         goto  Rotate
              00627
013D 0C16     00628 Send1   movlw LOW1BIT         ; Send a ‘1’ bit
013E 0976     00629         call  LineLow         ; hold the line low for 1/3rd of a Bit Cell
013F 0C32     00630         movlw HI_1BIT
0140 0983     00631         call  LineHi          ; let the line high for the rest of the Bit Cell
              00632
0141 0403     00633 Rotate  bcf    STATUS,C       ; Rotate out the MSB bit just sent from
0142 0360     00634         rlf    INDF,F         ; the Temporary Data Register
0143 02F7     00635         decfsz BitCntr,F      ; count down as bits are sent
0144 0B36     00636         goto   SndBits        ; loop until 8 bits are sent
0145 06B4     00637         btfsc  Flags1,F1Sent1 ; see whether all data has been sent
0146 0B4A     00638         goto   SndStop        ; if so, go send the Stop Bit
0147 05B4     00639         bsf    Flags1,F1Sent1 ; if not, set the Sent Flag,
0148 02A4     00640         incf   FSR,F          ; Then go prepare to send the next 8 bits,
0149 0B34     00641         goto   SetSend        ; and send the data from the next Data register
              00642
014A 0C38     00643 SndStop movlw LOW0BIT         ; Send a ‘0’ bit to the Host
014B 0976     00644         call  LineLow
014C 0C14     00645         movlw HI_0BIT
014D 0574     00646         bsf   Flags1,F1Stop   ; indicate to LineHi that this is the Stop
014E 0983     00647         call  LineHi         ; Bit let the line go high for 2/3rd of a Bit Cell
014F 04F4     00648         bcf   Flags1,F1Cllsn  ; a Collision did not occur, clear the flag
0150 0415     00649         bcf   Flags2,F2Srq    ; an Srq is no longer needed
0151 04D5     00650         bcf   Flags2,F2DSend  ; the Data has been sent
0152 0634     00651         btfsc Flags1,F1Reg3   ; If current Data Reg. is 3, don’t allow
0153 0BC7     00652         goto  RunTsk2         ; Reg. 3 to be cleared (or at least the 1st 2
0154 0213        00653         movf  RAMaddr,W      ; bytes) clear the Data Registers from which
0155 0024        00654         movwf FSR            ; the Data was sent via temporary registers
0156 0060        00655         clrf  INDF           ; Clear the registers holding the originalData
0157 02A4        00656         incf  FSR,F          ; which was just sent via the temporary regs.
0158 0060        00657         clrf  INDF           ; Go setup to run the 2nd Application Task for
0159 0BC7        00658         goto  RunTsk2        ; the time between the end of data sent, and
                 00659                              ; the beginning of the next Attention Signal
                 00660
015A 0634        00661 Collisn btfsc Flags1,F1Reg3  ; if there was a collision during a Talk
015B 05F4        00662         bsf   Flags1,F1Cllsn ; Reg. 3 Command, then set the Collision
015C 0AAB        00663         goto  AttnSig        ; Flag, otherwise, just abort to Attn Signal
                 00664
                 00665 ;**********************************************************************
                 00666
                 00667 ;*** RECEIVE DATA FROM THE HOST *** (Listen; calls Tlt, GetBit) ***
                 00668 ; Get the Tlt Signal (Stop to Start Time)
                 00669 ; Tlt recognizes the beginning of the Start Bit
                 00670 ; Load indirect address of temporary Data register
                 00671 ; Get the rest of the Start Bit
                 00672 ; Receive the first Data byte from the Host into the temporary Data
                 00673 ; register by calling GetBit - GetBit uses indirect address
                 00674 ; Set indirect address to 2nd temporary Data register
                 00675 ; Receive the second Data byte from the Host into the temporary Data
                 00676 ; register And then receive the Data Stop Bit if the
                 00677 ; data was not for Reg. 3, move the Data now stored in the temporary
                 00678 ; Data registers into the RAM locations of the Data register designated
                 00679 ; in RAMaddr, and go run the 2nd Application Task.
                 00680 ; if the data was for Reg. 3, go interpret what the Data Command was
                 00681 ; and take appropriate action.
                 00682
015D 0594        00683 Listen  bsf   Flags1,F1Lstn ; Set Listen Flag to tell Tlt (Stop to Start Time)
015E 094C        00684         call   Tlt          ; to look for the beginning of the Start Bit
015F 0C10        00685         movlw  TmpReg1      ; receive bits into temporary registers
0160 0024        00686         movwf  FSR          ; use indirect addressing to store received Data
0161 0060        00687         clrf   INDF         ; in temporary registers
0162 02A4        00688         incf   FSR,F        ;
0163 0060        00689         clrf   INDF         ; clear any data currently in temporary registers
0164 00E4        00690         decf   FSR,F        ;
0165 0C32        00691         movlw  BIT_TST      ; load the TimeVariable to look for the rest of
0166 0039        00692         movwf  TimeVar      ; the Start Bit
0167 0403        00693         bcf    STATUS,C     ; clear the Carry bit so it doesn’t wrap around
0168 090C        00694         call   Get_Bit      ; get the rest of the Start bit
0169 0700        00695         btfss  INDF,LSB     ; it should be a ‘1’ bit
016A 0AAB        00696         goto   AttnSig      ; if not, abort to the Attn Signal
016B 0400        00697         bcf    INDF,LSB     ; don’t let the Start Bit be the 1st bit of Data
016C 0C08        00698 SetRecv movlw  BYTE         ; setup to receive 8 bits at a time into the reg.
016D 0037        00699         movwf  BitCntr      ; count down as bits come in
016E 0C48        00700 RcvData movlw  MAX_BIT      ; get & rotate a 1 or 0 bit into Data Reg., and
016F 0039        00701         movwf  TimeVar      ; see if MAX_BIT time is exceeded & if so, abort
0170 0403        00702         bcf    STATUS,C     ; clear Carry bit so it doesn’t wrap around
0171 0360        00703         rlf    INDF,F       ; rotate the bit into the Register (the 1st
0172 090C        00704         call   Get_Bit      ; rotation doesn’t count)
0173 02F7        00705         decfsz BitCntr,F    ;decrement the counter each time a bit is
0174 0B6E        00706         goto   RcvData      ;received loop until 8 bits are received
0175 06D4        00707         btfsc  Flags1,F1Rcvd1  ; see whether the 2nd Data byte was just
0176 0B7A        00708         goto   RcvStop      ;received if so, go get the Stop Bit
0177 05D4        00709         bsf    Flags1,F1Rcvd1  ; if not, set the Received-1st-Byte Flag,
0178 02A4        00710         incf   FSR,F        ;increment FSR to receive 2nd Byte of the Data
0179 0B6C        00711         goto   SetRecv      ;Reg. & go prepare to receive the next byte
                 00712
017A 0C48        00713 RcvStop movlw  MAX_BIT      ;Get the ‘0’ Stop Bit
017B 0039        00714         movwf  TimeVar      ;
017C 0201        00715 RecvTmp movf   TMR0,W       ;Check the time, then check if the line went high
017D 0099        00716         subwf  TimeVar,W    ;See if more than MAX_BIT usecs have passed
017E 0703        00717         btfss  STATUS,C     ;if so, abort to Attn Signal
017F 0AAB        00718         goto   AttnSig      ;
```

```
0180 0705    00719         btfss  PORTA,ADB     ; if not, check whether the line went high
0181 0B7C    00720         goto   RecvTmp       ; if still low, loop to check the time again
0182 0C32    00721         movlw  BIT_TST       ; if high, make sure the Stop Bit was ‘0’
0183 0039    00722         movwf  TimeVar       ; if the time was < BIT_TST, abort to
0184 0201    00723         movf   TMR0,W        ; the Attn Signal
0185 0099    00724         subwf  TimeVar,W     ; if the time was > BIT_TST, the ‘0’ Stop
0186 0603    00725         btfsc  STATUS,C      ; Bit was received
0187 0AAB    00726         goto   AttnSig       ; clear TMR0 so second Task may use idle time
             00727
0188 0061    00728 RcvdDat clrf   TMR0          ; Move Data to registers (unless for Reg 3.)
0189 0634    00729         btfsc  Flags1,F1Reg3 ; see if Data was received for Register 3,
018A 0B94    00730         goto   DataChk       ; if so, go interpret the Listen Reg. 3
018B 0213    00731         movf   RAMaddr,W     ; Command if not, move the received Data bytes
018C 0024    00732         movwf  FSR           ; to their indicated registers using indirect
018D 0210    00733         movf   TmpReg1,W     ; address,
018E 0020    00734         movwf  INDF
018F 02A4    00735         incf   FSR,F
0190 0211    00736         movf   TmpReg2,W
0191 0020    00737         movwf  INDF
0192 05B5    00738         bsf    Flags2,F2DRcvd; set the Data-has-been-received flag,
0193 0BC7    00739         goto   RunTsk2       ; and go prepare to run the 2nd Application Task
             00740
             00741 ;*************************************************************************
             00742
             00743 ;* INTERPRET THE LISTEN REG. 3 COMMAND SENT BY THE HOST *** (DataChk) *
             00744 ; This interprets the Data received for Register 3 as one of the
             00745 ; following Commands and runs the corresponding routine:
             00746 ;
             00747 ; Mask the Data Command received using the following Constants passed
             00748 ; to the IntData (Interpret Data Command) macro:
             00749 ; SELFTST (FF) - the Device is instructed to do a Self-Test
             00750 ; LISTEN1 (00) - unconditionally change Device Address and/or Status bits
             00751 ; LISTEN2 (FE) - change only the Device Address, and only change it
             00752 ;                if the Device Address is marked as movable
             00753 ; DEV_ACT (FD) - change Device Address only if the Device Activator is
             00754 ;                pressed (as defined in Device specification)
             00755
             00756 DataChk IntData SELFTST,SlfTest ; see if Data Command is for Self Test
0194 0211        M         movf    TmpReg2,W       ; interprets the Data Command received by
0195 0FF         M         xorlw   SELFTST         ; comparing the 2nd byte to a Data
0196 0643        M         btfsc   STATUS,Z        ; Command constant
0197 0BA7        M         goto    SlfTest         ; it then goes to the appropriate routine
             00757         IntData LISTEN1,UpDat3a ; update bits Address and Status Bits (8
0198 0211        M         movf    TmpReg2,W       ; to 13) interprets the Data Command
0199 0F00        M         xorlw   LISTEN1         ; received by comparing the 2nd byte to a Data
019A 0643        M         btfsc   STATUS,Z        ; Command constant
019B 0BA9        M         goto    UpDat3a         ; it then goes to the appropriate routine
             00758         IntData LISTEN2,NewAddr ; change the Device Address (Bits 8 to 12)
019C 0211        M         movf    TmpReg2,W       ; interprets the Data Command received by
019D 0FFE        M         xorlw   LISTEN2         ; comparing the 2nd byte to a Data
019E 0643        M         btfsc   STATUS,Z        ; Command constant
019F 0BAF        M         goto    NewAddr         ; it then goes to the appropriate routine
             00759         IntData DEV_ACT,DevActv ; change the Device Address if the Device
01A0 0211        M         movf    TmpReg2,W       ; interprets the Data Command received by
01A1 0FFD        M         xorlw   DEV_ACT         ; comparing the 2nd byte to a Data
01A2 0643        M         btfsc   STATUS,Z        ; Command constant
01A3 0BAD        M         goto    DevActv         ; it then goes to the appropriate routine
             00760                                 ; Activator was pressed
01A4 0211    00761         movf    TmpReg2,W    ; if none of these Commands were given, put
01A5 002F    00762         movwf   Reg3b        ; received Data into Reg. 3b as a new Device the
01A6 0BC7    00763         goto    RunTsk2      ; Handler ID and go prepare to run the 2nd Task
             00764
01A7 0575    00765 SlfTest bsf     Flags2,F2STest  ; Tell Device to do a Self-Test during 2nd
01A8 0BC7    00766         goto    RunTsk2         ; Task, and go prepare to run the 2nd Task
             00767
01A9 0210    00768 UpDat3a movf    TmpReg1,W       ; Unconditionally change the Device Address
```

```
                00769                             ; and/or the Status Bits of Reg. 3a
01AA 05C0       00770         bsf   W,ExpEvnt     ; NOTE: Exceptional Event should remain as
01AB 002E       00771         movwf Reg3a         ; set to a ‘1’ unless otherwise indicated
01AC 0BC7       00772         goto  RunTsk2       ; Go prepare to run the 2nd Application Task
                00773
01AD 0755       00774 DevActv btfss Flags2,F2DActv; if the Device Activator was NOT pressed,
01AE 0BC7       00775         goto  RunTsk2       ; go run the 2nd Application Task,
                00776                             ; if it was, change Device Address, if movable
01AF 06F4       00777 NewAddr btfsc Flags1,F1Cllsn; If a collison occurred during the last
01B0 0AAB       00778         goto  AttnSig       ; Talk Reg. 3, the Address was marked unmov
01B1 0210       00779         movf  TmpReg1,W     ; able, abort to the Attention Signal.
01B2 0F00       00780         xorlw FALSE         ;
01B3 0643       00781         btfsc STATUS,Z
01B4 0AAB       00782         goto  AttnSig
01B5 020E       00783         movf  Reg3a,W       ; Create the new Device Address by masking in
01B6 0EF0       00784         andlw HI_NIBL       ; the Address received by the host, not allowing
01B7 0031       00785         movwf TmpReg2       ;  the upper nibble Status Bits in Reg. 3a to
01B8 0210       00786         movf  TmpReg1,W     ;  be affected.
01B9 0E0F       00787         andlw LOW_NBL       ;
01BA 0111       00788         iorwf TmpReg2,W     ; NOTE: Exceptional Event should remain as
01BB 05C0       00789         bsf   W,ExpEvnt     ; set to a ‘1’ unless otherwise indicated
01BC 002E       00790         movwf Reg3a         ;   when the new Device Address is in place,
01BD 0BC7       00791         goto  RunTsk2       ; go prepare to run the 2nd Application Task
                00792
                00793 ;***********************************************************************
                00794 ;*** FLUSH THE REGISTER SPECIFIED BY THE COMMAND BYTE *** (Flush) ***
                00795
01BE 0213       00796 Flush   movf    RAMaddr,W   ; Clear the Data in the specified Register
01BF 0024       00797         movwf   FSR         ; use indirect address to clear the RAM
01C0 0060       00798         clrf    INDF        ; locations holding the Data
01C1 02A4       00799         incf    FSR,F
01C2 0060       00800         clrf    INDF
01C3 0BC7       00801         goto    RunTsk2
                00802
                00803 ;************************************************************************
                00804
01C4 0AAB       00805 Reserv1 goto    AttnSig     ; No action until Reserved Command 1 is defined
                00806
                00807 ;***********************************************************************
                00808
01C5 0AAB       00809 Reserv2 goto    AttnSig     ; No action until Reserved Command 2 is defined
                00810
                00811 ;***********************************************************************
                00812
01C6 0AAB       00813 Reserv3 goto    AttnSig     ; No action until Reserved Command 3 is defined
                00814
                00815 ;***********************************************************************
                00816
                00817 ;*** PUT THE CODE FOR OTHER APPLICATION HERE *** (RunTsk2, Task_2) ***
                00818
                00819 ;        bsf   Flags2,F2SFail ; code would go before here if a Self Test
                00820 ;        bcf   Flags2,F2SFail ; was performed and it failed or passed
                00821
01C7 0070       00822 RunTsk2 clrf  TmpReg1         ; When finished with Data interpretation,
01C8 0071       00823         clrf  TmpReg2         ; clear the temporary Data registers, and
01C9 0CE1       00824         movlw TSK2MAX         ; load Task 2 TimeVariable with amount allowed
01CA 003A       00825         movwf Tsk2Var         ; between end of Data and Attention Signal. If
                00826
01CB 0615       00827 Task_2  btfsc Flags2,F2Srq    ; the Srq Flag has not been cleared, then data
01CC 0BF7       00828         goto  AttnTst         ; must still be sent from 1st Service Request
01CD 0900       00829         call  PrScale         ; Turn on the TMR0 prescale for >250usec count
                00830
01CE 0675       00831 Tests   btfsc Flags2,F2STest  ; See if Key-Up transition codes should be
01CF 0BDC       00832         goto  LoadDat         ; sent
01D0 0635       00833         btfsc Flags2,Switch   ; Determine if the Switch has been
01D1 0BED       00834         goto  DBounce         ; de-bounced if not, go timeout
```

```
01D2 0626      00835         btfsc   PORTB,Switch  ; Check if Switch is pressed,
01D3 0BF3      00836         goto    Tsk2Tmp       ; if not, go timeout
01D4 05F5      00837         bsf     Flags2,F2DMore; data needs to be sent to the host
01D5 0515      00838         bsf     Flags2,F2Srq  ; and issue a Service Request
01D6 0535      00839         bsf     Flags2,Switch ; set the flag for de-bouncing switch
01D7 0506      00840         bsf     PORTB,LED     ; Turn on LED when Switch is pressed
01D8 0C08      00841         movlw   DEBOUNC
01D9 003B      00842         movwf   TmpCtr1
01DA 06D5      00843         btfsc   Flags2,F2DSend; The last Data was sent correctly if Talk
01DB 0BF3      00844         goto    Tsk2Tmp       ; cleared the DSend flag, if set, goto
               00845                               ; Attn Test to re-send Data
               00846
01DC 07F5      00847 LoadDat btfss   Flags2,F2DMore; If all the Data has been sent, DMore is
01DD 0BF3      00848         goto    Tsk2Tmp       ; clearif DMore is clear, go time out
01DE 0C38      00849         movlw   SHIFT         ; if DMore is set, Data remains to be sent
01DF 0028      00850         movwf   Reg0a         ; if not, load the Data bytes
01E0 0C12      00851         movlw   BANG
01E1 0029      00852         movwf   Reg0b
01E2 05D5      00853         bsf     Flags2,F2DSend; Data now needs to be sent to the host
01E3 0515      00854         bsf     Flags2,F2Srq  ; Until all data has been sent, Srq’s may
01E4 0675      00855         btfsc   Flags2,F2STest; be sent See if Key-Up Transition Codes
01E5 0BE8      00856         goto    KeyUp         ; should be sent if so, go set the bits
01E6 0575      00857         bsf     Flags2,F2STest; if not, set bit so they will be next time
01E7 0BED      00858         goto    DBounce       ;  and go debounce the switch
               00859
01E8 05E8      00860 KeyUp   bsf     Reg0a,07h     ; Set the 7th bit in each register to
01E9 05E9      00861         bsf     Reg0b,07h     ; indicate the Key is up
01EA 0475      00862         bcf     Flags2,F2STest; The Key-Up Transition Code bits have been
01EB 04F5      00863         bcf     Flags2,F2DMore; set All data will have been sent to the
01EC 0BED      00864         goto    DBounce       ; host after this transaction
               00865
01ED 0726      00866 DBounce btfss   PORTB,Switch  ; Check if Switch has been released,
01EE 0BF3      00867         goto    Tsk2Tmp       ; if not, go timeout
01EF 02FB      00868         decfsz  TmpCtr1,F     ; if so, start timed debounce of several
01F0 0BF3      00869         goto    Tsk2Tmp       ; millisecs. before switch is tested again
01F1 0406      00870         bcf     PORTB,LED     ; Turn off LED when Switch is released
01F2 0435      00871         bcf     Flags2,Switch ; clear de-bounce flag
               00872
01F3 0201      00873 Tsk2Tmp movf    TMR0,W      ; Check the time to see if more than the maximum
01F4 009A      00874         subwf   Tsk2Var,W   ; time limit has been exceeded
01F5 0603      00875         btfsc   STATUS,C    ; if so, go determine what part of Attn Signal
01F6 0BF3      00876         goto    Tsk2Tmp
               00877
01F7 0714      00878 AttnTst btfss   Flags1,F1Attn ; After this portion of the 2nd Task is
01F8 0AAB      00879         goto    AttnSig       ; complete,If 2nd Task is NOT run during
01F9 0414      00880         bcf     Flags1,F1Attn ; Attn Signal, go get the start of the Attn
01FA 0AC2      00881         goto    AttnMin       ; Signal otherwise, go get the rest of the
               00882                               ; Attn Signal
               00883 ;***********************************************************************
               00884
01FF           00885         ORG     PIC54
01FF 0A90      00886 RESETV  goto    Start
               00887
               00888     END
```

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0180 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
01C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX----X

All other memory blocks unused.

Program Memory Words Used:   508
Program Memory Words Free:   516


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```

Note the following details of the code protection feature on PICmicro® MCUs.

- The PICmicro family meets the specifications contained in the Microchip Data Sheet.
- Microchip believes that its family of PICmicro microcontrollers is one of the most secure products of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the PICmicro microcontroller in a manner outside the operating specifications contained in the data sheet. The person doing so may be engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable”.
- Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our product.

If you have any further questions about this matter, please contact the local sales office nearest to you.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip’s products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

Printed on recycled paper.

DNV Certification, Inc.
USA

ANSI ★ RAB
QMS
ACCREDITED

DNV

DNV MSC
The Netherlands
Accredited by the RvA

ISO 9001 / QS-9000
REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company’s quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip’s quality system for the design and manufacture of development systems is ISO 9001 certified.

MICROCHIP®

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200 Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: http://www.microchip.com

Rocky Mountain
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966 Fax: 480-792-7456

Atlanta
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston
2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848 Fax: 978-692-3821

Chicago
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

Detroit
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Kokomo
2767 S. Albright Road
Kokomo, Indiana 46902
Tel: 765-864-8360 Fax: 765-864-8387

Los Angeles
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto
6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699 Fax: 905-673-6509

ASIA/PACIFIC

Australia
Microchip Technology Australia Pty Ltd
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

China - Beijing
Microchip Technology Consulting (Shanghai)
Co., Ltd., Beijing Liaison Office
Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Chengdu
Microchip Technology Consulting (Shanghai)
Co., Ltd., Chengdu Liaison Office
Rm. 2401, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-6766200 Fax: 86-28-6766599

China - Fuzhou
Microchip Technology Consulting (Shanghai)
Co., Ltd., Fuzhou Liaison Office
Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506 Fax: 86-591-7503521

China - Shanghai
Microchip Technology Consulting (Shanghai)
Co., Ltd.
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

China - Shenzhen
Microchip Technology Consulting (Shanghai)
Co., Ltd., Shenzhen Liaison Office
Rm. 1315, 13/F, Shenzhen Kerry Centre,
Renminnan Lu
Shenzhen 518001, China
Tel: 86-755-2350361 Fax: 86-755-2366086

Hong Kong
Microchip Technology Hongkong Ltd.
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

India
Microchip Technology Inc.
India Liaison Office
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan
Microchip Technology Japan K.K.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan
Microchip Technology Taiwan
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Denmark
Microchip Technology Nordic ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Microchip Technology GmbH
Gustav-Heinemann Ring 125
D-81739 Munich, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom
Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5869 Fax: 44-118 921-5820

01/18/02