Multimaster I²C routines for byte-oriented I²C interfaces — AN435

DESCRIPTION

This application note consists of a set of drivers to allow easy use of multimaster I²C on Philips microcontrollers that have the byte oriented I²C interface. Some devices that include this version of the I²C interface are the 8XC552, 8XC562, 8XC652, 8XC654, 8XCL410, 8XCL580, and the 8XCL781. This program is used as an I²C driver which communicates with the user's main program using a simple macro language.

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

```
$Title(I2C Byte Oriented Software Driver)
$Date(04/22/92)
;
;*****************************************************************************
;I2C Byte Oriented System Driver.
;Written by Joe Brandolino, FAE, Etobicoke Sales Office.
;Region 47 (Canada).
;*****************************************************************************
;
;DESCRIPTION:
;============
;IIC_OS.ASM contains a complete multimaster I2C driver for the byte
;oriented Philips microcontrollers. To date, the list of byte
;oriented 80C51 derivative microcontrollers includes:
;
;                             – 8XC552
;                             – 8XC562
;                             – 8XC652
;                             – 8XC654
;                             – 8XCL410
;                             – 8XCL580
;                             – 8XCL781
;
;IIC_OS was written for Philips customers who do not want to spend
;the many hours required to develop a complete multimaster IIC driver.  This
;program is used as an IIC driver which communicates with the main program
;using a simple macro language.
;
;The comments in this listing assume that the reader has a basic knowledge of
;the 80C51 family, and is familiar with IIC basics.  This program has been
;tested as thoroughly as time permitted; however, Philips cannot
;guarantee that this IIC driver is flawless in all applications.
;
;The comment text fields in this file use a consistent method of highlighting
;the various parameters of the software.  All constants (EQUates), registers,
;bits and other bytes are surrounded by ’ ’ in the comment text. All routines,
;labels, procedures and file names are surrounded by ” ” in the comment text.
;Generally speaking, all 8051 mnemonics are in UPPERCASE, all variable names
;and labels are in LOWERCASE or mixed case.  The terms IIC and I2C are used
;interchangeably, and both mean Inter–Integrated Circuit.
;
;–––NOTE–––NOTE–––NOTE–––NOTE–––NOTE–––NOTE–––NOTE–––NOTE–––NOTE–––NOTE–––NOTE
;
;To incorporate this program into your main program, place it somewhere in
;your source text file by including the following text:
;
;         $include(mod552)      ;include the desired processor descriptor file
;         $include(iic_os.asm)                          ;include this program
;
;Since this program has a ’CSEG AT... definition for the IIC interrupt vector,
;it is probably best to place it in your program where all the other interrupt
;vector directives reside so that assembly synchronization errors do not
;occur.
;
```

```
;You must also ensure that the data bytes used by this program do not
;conflict with those in your main program.  Don't forget to initialize
;the IIC control registers and the interrupt registers, etc.  For example:
;
;INIT:        .
;             .
;             .
;        MOV     P1,#0                                                ;init P1
;        MOV     P1,#0FFH
;        MOV     IEN0,#10100000B                         ;enable IIC interrupt
;        MOV     S1CON,#ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0
;        MOV     S1ADR,#Own_adrs OR general_enable     ;enable slave/general mode
;        MOV     IIC_status,#status_OK                  ;init system status byte
;        CLR     IIC_failure                                    ;init status bit
;             .
;             .
;
;This driver uses DATA space bytes from decimal address 48 to 78 (16 of these
;bytes are for slave mode receive and transmit buffers – this space can be re-used
;if not required). Bit space addresses used are from 0 to 9 decimal. These
;addresses can be moved to any convenient location in your system. If the
;driver is used as is, then start your DATA space definitions at 'DATA_start'
;decimal (i.e. DSEG AT DATA_start) and your BIT space definitions at
;'BITS_start' (i.e. BSEG AT BITS_start). There are no register banks used per
;se – all registers required are pushed onto the stack if used.
;To interface to this IIC Driver, the user need not understand all the details
;of the program - only the following registers must be understood:
;
;          'ICC_Command_file_adrs'          – used in every command file
;          'indirect_adrs'                  – used only with 'indirect-'option
;          'indirect_count'                 – used only with 'indirect-'option
;          'single_data'                    – used only with 'singleD_' option
;          'Slave_in' buffer (if required)  – used only in multimaster systems
;          'Slave_out' buffer (if required) – used only in multimaster systems
;          'IIC_failure' (BIT)              – set if command file was kaput
;          'IIC_status'                     – holds final status of session
;
;Additionally, there is a command file structure (the command file is a
;list of commands that "IIC_OS" will execute) which the user must conform to.
;The list of commands includes:
;
;          'ioD_'         _ target DATA space for I/O transfers
;          'ioC_'         _ target CODE space for I/O transfers
;          'ioX_'         _ target XDATA space for I/O transfers
;          'immediate_'   _ used to output 1 byte from command file stream
;          'call_'        – used to call a subroutine between repeated starts
;          'indirect_'    – gets I/O address and count from 'indirect_ registers
;          'singleD_'     – gets 1 byte of I/O data from 'single_data'
;
;          'iicend'       – last byte of a command file
;          'iicwritemask' – OR with slave address to indicate a write operation
;          'iicreadmask'  – OR with slave address to indicate a read operation
;
;The command file structure is explained in detail below.
;–––NOTE–––NOTE–––NOTE–––NOTE–––NOTE–––NOTE–––NOTE–––NOTE–––NOTE–––NOTE–––NOTE

;Multimaster systems are very specific to the system design, and therefore,
;very difficult to make generic.  Every multimaster system will have a
;different protocol for how many (and which) bytes to send/receive when the
;master is addressed as a Slave Receiver or Slave Transmitter.  For this
;reason, this program implements the multimaster scenario very simply –
;if the micro running this program is addressed as a slave, it will read
;'SLVbytes_in' number of data bytes or write 'SLVbytes_out' number of data
;bytes (depending on what the calling master requests).  The target data
;buffer in these cases are the 'Slave_in' buffer and the 'Slave_out' buffer.
;
```

Multimaster I²C routines for byte-oriented I²C interfaces AN435

```
;The user can make the size of these slave input/output buffers (and the
;corresponding equates 'SLVbytes_in' and 'SLVbytes_out') as large as
;required.  The calling Master can terminate the slave session at any number
;of data bytes sent or received by providing a stop or a not acknowledge.
;
;IIC_OS, when integrated into the user's system, will require 15 DATA bytes
;(mapped anywhere in the internal DATA memory space), and one bit-addressable
;byte.  About 600 bytes of code-space memory are used.
;
;The user of this program need not concern himself with the bit or byte level
;operation of the IIC hardware - this program takes care of all IIC registers,
;and checks for all collisions, arbitration lost scenarios, bus errors, etc.
;A command list consisting of a limited number of simple macro commands is
;set-up by the user, and this driver uses that list of commands to perform
;the desired IIC operations.
;
;The user loads the 'IIC_Command_File_adrs' (2 byte) register with
;the address of the sequence of IIC operations desired.  Once this register
;is loaded, the "WAIT_IIC_Data", "WAIT_IIC_Xdata", or "WAIT_IIC_Code" routine
;is called, depending on which data space the command file list resides.
;
;"WAIT_IIC" starts the IIC interrupt service routine by setting the 'STA'
;(IIC start) bit.  Then the "IIC_VECTOR" routine is entered after every
;significant IIC event (this occurs because of the IIC hardware in the
;microcontroller).  The interrupt service routine takes care of setting the
;IIC hardware registers and checking for collisions and stepping through the
;IIC command file. "WAIT_IIC" also does a timeout feature for the IIC system.
;
;The IIC operations to be performed are stored sequentially starting at the
;address specified by the 'IIC_Command_File_adrs' and in the memory space
;designated by 'Command?adrs?space' (the later register is loaded with the
;appropriate memory space code through the call to the "WAIT_IIC_xxxx"
;routine).  IIC operations include:
;
;          1) sending or receiving any number of bytes from 1 to 255
;             into any valid address space
;          2) repeated start automatically performed so multiple
;             slaves can be communicated with in one call
;          3) call subroutines between repeated start conditions directly
;             from the IIC command file list (i.e. transparent to the
;             calling routine).
;
;The IIC Command File must be constructed so that it conforms to the IIC
;driver system format.  This format is very simple and is outlined later.
;The IIC Command File is built from 1 to any number of blocks.  Each block
;is from 2 to 8 bytes long, depending on what functions must be performed.
;There are only eight types of blocks, indicated by the options in the format
;below and briefly explained here:
;
;option 1 FUNCTION:  send/receive bytes to/from slave from/to any memory space
;         # BYTES IN THIS COMMAND FILE BLOCK:                                5
;         NUMBER OF BYTES TO SEND/RECEIVE:        get number from command file
;         ADDRESS FOR DATA DERIVED FROM:                          command file
;         OTHER FUNCTIONS:                                                none
;         COMMENTS: Option 1 is useful for sending or receiving any specified
;                   number of data bytes to/from the specified slave.  Every
;                   required piece of information is stored in the command
;                   file - that is, the address of the slave + read/write bit,
;                   the number of bytes to send or receive, and the address
;                   to send from or receive to.  Recall that the address space
;                   is always specified in the command file.
;
;option 2 FUNCTION:                   send one byte to slave from command file
;         # BYTES IN THIS COMMAND FILE BLOCK:                                3
;         NUMBER OF BYTES TO SEND/RECEIVE:                                   1
;         ADDRESS FOR DATA DERIVED FROM:  data read directly from command file
;         OTHER FUNCTIONS:                                                none
```

```
;         COMMENTS: Option 2 is used to send exactly one byte to an addressed
;                   slave.  The byte is fixed and is stored in the command
;                   file itself; this method reduces command file bytes since
;                   no specification for data address or number of data bytes
;                   is necessary.  Option 2 provides a simple means of setting
;                   the sub–address in IIC memory devices.
;
;option 3 FUNCTION:  send/receive bytes to/from slave from/to any memory space
;         # BYTES IN THIS COMMAND FILE BLOCK:                                2
;         NUMBER OF BYTES TO SEND/RECEIVE:    get number from ’indirect_count’
;         ADDRESS FOR DATA DERIVED FROM:   get address from ’indirect_address’
;         OTHER FUNCTIONS:                                                none
;         COMMENTS: Option 3 assumes that the calling program has set–up the
;                   ’indirect_count’ register with the number of bytes to be
;                   sent or received, and the ’indirect_address’ with the
;                   address of the bytes to be sent or received.  The data
;                   space targeted is specified in the command file as
;                   usual.
;
;option 4 FUNCTION:  send/receive bytes to/from slave from/to any memory space
;         # BYTES IN THIS COMMAND FILE BLOCK:                                7
;         NUMBER OF BYTES TO SEND/RECEIVE:        get number from command file
;         ADDRESS FOR DATA DERIVED FROM:         get address from command file
;         OTHER FUNCTIONS:              CALL subroutine listed in command file
;         COMMENTS: Option 4 is identical to Option 1, except that a
;                   subroutine (whose address is specified in the command
;                   file) is called after the data transfer is complete.
;
;option 5 FUNCTION:                   send one byte to slave from command file
;         # BYTES IN THIS COMMAND FILE BLOCK:                                5
;         NUMBER OF BYTES TO SEND/RECEIVE:                                   1
;         ADDRESS FOR DATA DERIVED FROM:  data read directly from command file
;         OTHER FUNCTIONS:              CALL subroutine listed in command file
;         COMMENTS: Option 5 is identical to Option 2, except that a
;                   subroutine (whose address is specified in the command
;                   file) is called after the data transfer is complete.
;
;option 6 FUNCTION:  send/receive bytes to/from slave from/to any memory space
;         # BYTES IN THIS COMMAND FILE BLOCK:                                4
;         NUMBER OF BYTES TO SEND/RECEIVE:    get number from ’indirect_count’
;         ADDRESS FOR DATA DERIVED FROM:   get address from ’indirect_address’
;         OTHER FUNCTIONS:              CALL subroutine listed in command file
;         COMMENTS: Option 6 is identical to Option 3, except that a
;                   subroutine (whose address is specified in the command
;                   file) is called after the data transfer is complete.
;
;option 7 FUNCTION:  send/receive one byte to/from slave from/to ’single_data’
;         # BYTES IN THIS COMMAND FILE BLOCK:                                2
;         NUMBER OF BYTES TO SEND/RECEIVE:                                   1
;         ADDRESS FOR DATA DERIVED FROM:            ’single_data’ is addressed
;         OTHER FUNCTIONS:                                                none
;         COMMENTS: Option 7 allows the user to send or receive exactly one
;                   data byte to/from the slave using the ’single_data’
;                   register as the target.  The calling routine will have to
;                   write the desired data into the ’single_data’ register if
;                   a write operation to the slave is desired. This option
;                   requires very few command file bytes since count and
;                   address information are not needed.
;
;option 8 FUNCTION:  send/receive one byte to/from slave from/to ’single_data’
;         # BYTES IN THIS COMMAND FILE BLOCK:                                4
;         NUMBER OF BYTES TO SEND/RECEIVE:                                   1
;         ADDRESS FOR DATA DERIVED FROM:            ’single_data’ is addressed
;         OTHER FUNCTIONS:              CALL subroutine listed in command file
;         COMMENTS: Option 8 is identical to Option 7, except that a
;                   subroutine (whose address is specified in the command
;                   file) is called after the data transfer is complete.
```

Multimaster I^{2}C routines for byte-oriented I^{2}C interfaces AN435

```
;
;
;IIC_OS Command File Block Format:
;*********************************
;NOTE: uppercase 'OR' = logical or function in the following text.
;
;byte # 1 = 7 bit address of slave OR 'iicreadmask' (or 'iicwritemask')
;========
;
;byte # 2 :
;==========
;  option 1 = address space code (which memory space transmit data is read
;                                  from or which memory space receive data is
;                                  written to is specified by an "address
;                                  space code" - see EQUates in main program.)
;
;  option 2 = 'immediate_' control code
;              ('immediate_'  indicates that the next byte is the actual data
;                              to be transmitted.  This of course is only valid
;                              when writing a byte to a slave.  This control
;                              code will save the bytes of information required
;                              to specify the address of the data to be
;                              transmitted.  It is a very handy and
;                              efficient mechanism for setting-up the read or
;                              write address in an I2C memory device.)
;
;  option 3 = address space code OR 'indirect_' control code
;             ('indirect_' indicates that the address for the bytes to be
;                           transmitted or received is not in the command file
;                           but is contained in the IIC_OS register
;                           'indirect_adrs'; also, the number of bytes to be
;                           transmitted or received is contained in the IIC_OS
;                           register 'indirect_count'.  The calling routine must
;                           preload these registers, or they must be correctly
;                           loaded from a previous "call_" initiated in the
;                           command file stream.)
;
;  option 4 = address space code OR 'call_' control code
;             ('call_' indicates that the address of the subroutine to be
;                       called after the present IIC transmission or reception
;                       is complete is contained in the bytes following.)
;
;  option 5 = 'immediate_' control code OR 'call_' control code
;
;  option 6 = address space code OR 'indirect_'  OR 'call_'
;
;  option 7 = 'singleD_'
;             ('singleD_' indicates that one byte is to be read/written, and
;                          that the target byte to be read/written is the IIC_OS
;                          byte called 'single_data'.)
;
;  option 8 = 'singleD_' OR 'call_'
;
;byte # 3 :
;==========
;  option 1 = number of bytes to be transmitted or received (1 to 255)
;
;  option 2 = the data to be transmitted (the 'immediate_' control code was
;             used in byte # 2)
;
;  option 3 = 'iicend' control code to end session, or next block's byte #1
;
;  option 4 = same as option 1
;
;  option 5 = same as option 2
;
```

```
;  option 6 = low address of subroutine to be called after this transmit or
;             receive session
;
;  option 7 = same as option 3
;
;  option 8 = same as option 6
;
;byte # 4 :
;==========
;  option 1 = low address of data to be transmitted or received
;
;  option 2 = ’iicend’ control code to end session, or next block’s byte #1
;
;  option 3 = Not Applicable
;
;  option 4 = same as option 1
;
;  option 5 = low address of subroutine to be called after this transmit or
;             receive session
;
;  option 6 = high address of subroutine to be called after this transmit or
;             receive session
;
;  option 7 = Not Applicable
;
;  option 8 = same as option 6
;
;byte # 5 :
;==========
;  option 1 = high address of data to be transmitted or received
;             (’iicend’ if target memory space is DATA)
;
;  option 2 = Not Applicable
;
;  option 3 = Not Applicable
;
;  option 4 = same as option 1
;
;  option 5 = high address of subroutine to be called after this transmit or
;             receive session
;
;  option 6 = ’iicend’ control code to end session, or next block’s byte #1
;
;  option 7 = Not Applicable
;
;  option 8 = same as option 6
;
;byte # 6 :
;==========
;  option 1 = ’iicend’ control code to end session, or next block’s byte #1
;             (Not Applicable if target memory space is DATA)
;
;  option 2 = Not Applicable
;
;  option 3 = Not Applicable
;
;  option 4 = low address of subroutine to be called after this transmit or
;             receive session
;
;  option 5 = ’iicend’ control code to end session, or next block’s byte #1
;
;  option 6 = Not Applicable
;
;  option 7 = Not Applicable
;
;  option 8 = Not Applicable
;
```

```
;byte # 7 :
;==========
;  option 1 = Not Applicable
;
;  option 2 = Not Applicable
;
;  option 3 = Not Applicable
;
;  option 4 = high address of subroutine to be called after this transmit or
;             receive session
;
;  option 5 = Not Applicable
;
;  option 6 = Not Applicable
;
;  option 7 = Not Applicable
;
;  option 8 = Not Applicable
;
;byte # 8 :
;==========
;  option 1 = Not Applicable
;
;  option 2 = Not Applicable
;
;  option 3 = Not Applicable
;
;  option 4 = 'iicend' control code to end session, or next block's byte #1
;
;  option 5 = Not Applicable
;
;  option 6 = Not Applicable
;
;  option 7 = Not Applicable
;
;  option 8 = Not Applicable
;
;To efficiently use this system, several of these blocks can be put together.
;In fact, there is no limit on the number of blocks allowed.
;This IIC_OS lends itself very nicely to complex IIC requirements.  The
;following examples will illustrate the usefulness of this program.
;
;EXAMPLES
;********
;The following examples are samples for each option.  There are so many
;variations that only one version for options 1 – 4 are presented.  The code
;fragment "PROGRAM" is simply the part of the code that sets up and calls
;the "WAIT_IIC" program which waits for the execution of the command file.
;The "Optionx_file" code fragments are the code space command files.  All
;of the examples show the command file residing in the code space, but they
;could just as easily have been loaded into the DATA or XDATA spaces.
;
;It should be noted that "IIC_OS" will process a command file until an
;'iicend' character is encountered – after which, a STOP condition will be
;implemented.  This means that the master can keep possession of the bus
;for as long as it has to.
;
;It is assumed that all the slave addresses and other EQUates have been
;defined in the program previously.
;
;EXAMPLE Option 1
;++++++++++++++++
;PROGRAM:
;        MOV     IIC_Command_File_adrs,#LOW(Option1_file)     ;load address of
;        MOV     IIC_Command_File_adrs+1,#HIGH(Option1_file)     ;command file
;        CALL    WAIT_IIC_Code         ;call program to wait for IIC execution
;        JMP     MORE_PROGRAM
```

```
;        ;
;        ;Notice the block structure of the command file.  Each block has
;        ;been spaced to accentuate this structure.
;        ;'Option1_file' tells the IIC_OS (called through "WAIT_IIC_Code") to
;        ;read 5 bytes of data in from 'slave1' and store the bytes in the
;        ;DATA space starting at location 'iic_input_data'; after this input
;        ;is done, 'slave2' has 3 bytes written to it from the DATA space
;        ;starting at address 'iic_input_data'.
;        ;Both blocks below are option 1 types, but the first is a read
;        ;and the second is a write.
;        ;
; Option1_file:
;        DB      slave1_address OR iicreadmask      ;slave1 address + read bit
;        DB      ioD_                              ;indicate DATA space target
;        DB      5                                   ;indicate number of bytes
;        DB      iic_input_data                 ;start address of target bytes
;
;        DB      slave2_address OR iicwritemask    ;slave2 address + write bit
;        DB      ioD_                              ;indicate DATA space target
;        DB      3                                   ;indicate number of bytes
;        DB      iic_input_data                 ;start address of target bytes
;
;        DB      iicend                                      ;end of iic session
;
;MORE_PROGRAM:
;        continue with program
;
;EXAMPLE Option 2
;++++++++++++++++
;PROGRAM:
;        MOV     IIC_Command_File_adrs,#LOW(Option2_file)     ;load address of
;        MOV     IIC_Command_File_adrs+1,#HIGH(Option2_file)     ;command file
;        CALL    WAIT_IIC_Code         ;call program to wait for IIC execution
;        JMP     MORE_PROGRAM
;        ;
;        ;Notice the block structure of the command file.  Each block has
;        ;been spaced to accentuate this structure.
;        ;'Option2_file' tells the IIC_OS (called through "WAIT_IIC_Code") to
;        ;write one byte of data to the addressed slave – the data is present
;        ;in the command file.  This action takes 3 bytes in the command file.
;        ;In this example, the address for a memory location in an IIC memory
;        ;peripheral will be set and the following block does an option 1
;        ;type of input.
;        ;
;Option2_file:
;        DB      slave_address OR iicwritemask      ;slave address + write bit
;        DB      immediate_                          ;send out next byte only
;        DB      1                                      ;data byte to be sent
;                                                    ;(end of option 2 block)
;        DB      slave_address OR iicreadmask         ;slave address + read bit
;        DB      ioD_                          ;memory space where bytes go to
;        DB      6                                ;number of bytes to be read
;        DB      iic_input_data + 1                    ;address of input target
;
;        DB      iicend
;
;MORE_PROGRAM:
;        continue with program
;
;EXAMPLE Option 3
;++++++++++++++++
;PROGRAM:
;        MOV     indirect_adrs,#LOW(input_data1)
;        MOV     indirect_adrs+1,#HIGH(input_data1)
;        MOV     indirect_count,#6
;        MOV     A,decision
;        JZ      PROGRAM_10
```

Multimaster I²C routines for byte-oriented I²C interfaces AN435

```
;         MOV     indirect_adrs,#LOW(input_data2)
;         MOV     indirect_adrs+1,#HIGH(input_data2)
;         MOV     indirect_count,#3
;
;PROGRAM_10:
;         MOV     IIC_Command_File_adrs,#LOW(Option3_file)     ;load address of
;         MOV     IIC_Command_File_adrs+1,#HIGH(Option3_file)     ;command file
;         CALL    WAIT_IIC_Code         ;call program to wait for IIC execution
;         JMP     MORE_PROGRAM
;
;         ;
;         ;Notice the block structure of the command file.  Each block has
;         ;been spaced to accentuate this structure.
;         ;'Option3_file' tells the IIC_OS (called through "WAIT_IIC_Code") to
;         ;read 'indirect_count' number of bytes into external ram space
;         ;starting at address 'indirect_adrs'.  In the body of "PROGRAM"
;         ;the 'indirect_ registers are loaded based on a decision.  In this
;         ;case, if the data byte 'decision' is zero, 6 bytes of data are
;         ;read from the slave and placed in external ram starting at the
;         ;address 'input_data1'; if 'decision' is not zero, three bytes of
;         ;data are read from the slave and placed in external ram starting
;         ;at address 'input_data2'.
;         ;
;Option3_file:
;         DB      slave_address OR iicreadmask        ;slave address + read bit
;         DB      ioX_ OR indirect_            ;read from external ram area and
;                                                      ;use indirect registers
;         DB      iicend
;
;MORE_PROGRAM:
;         continue with program
;
;EXAMPLE Option 4
;++++++++++++++++
;PROGRAM:
;         MOV     IIC_Command_File_adrs,#LOW(Option4_file)     ;load address of
;         MOV     IIC_Command_File_adrs+1,#HIGH(Option4_file)     ;command file
;         CALL    WAIT_IIC_Code              ;call program to wait for IIC execution
;         JMP     MORE_PROGRAM
;         ;
;         ;Notice the block structure of the command file.  Each block has
;         ;been spaced to accentuate this structure.
;         ;'Option4_file' tells the IIC_OS (called through "WAIT_IIC_Code") to
;         ;read in 4 bytes from slave1 into data area 'iic_input_data' then
;         ;make a call to the subroutine "op4_sub", then output 1 byte of
;         ;data from 'single_data' to slave2.  The output data was
;         ;manipulated by the called subroutine.  All this occurred without
;         ;a stop condition being generated, so the bus was retained for
;         ;the entire period of time.
;         ;
;Option4_file:
;         DB      slave1_address OR iicreadmask       ;slave address + read bit
;         DB      ioD_  OR  call_               ;indicate DATA space then call
;         DB      4                                  ;indicate number of bytes
;         DB      iic_input_data                ;start address of target bytes
;         DB      LOW(op4_sub)                       ;address of routine to be
;         DB      HIGH(op4_sub)                      ;executed after read done
;
;         DB      slave2_address OR iicwritemask    ;slave2 address + write bit
;         DB      singleD_                        ;indicate 'single_data' to be
;                                                  ;output (see option 7)
;         DB      iicend                                   ;end of iic session
;         ;
;         ;Subroutine "op4_sub" adds the first 4 bytes of 'iic_input_data'
;         ;and puts answer into 'single_data'.
;         ;
```

```
;op4_sub:
;        MOV     R0,#iic_input_data
;        MOV     R1,#4
;        CLR     A
;op4_loop:
;        ADD     A,@R0
;        INC     R0
;        DJNZ    R1,op4_loop
;        MOV     single_data,A
;        RET
;
;MORE_PROGRAM:
;        continue with program
;
;EXAMPLE Option 7
;++++++++++++++++
;PROGRAM:
;        MOV     IIC_Command_File_adrs,#LOW(Option7_file)     ;load address of
;        MOV     IIC_Command_File_adrs+1,#HIGH(Option7_file)     ;command file
;        CALL    WAIT_IIC              ;call program to wait for IIC execution
;        JMP     MORE_PROGRAM
;        ;
;        ;Notice the block structure of the command file.  Each block has
;        ;been spaced to accentuate this structure.
;        ;'Option7_file' tells the IIC_OS (called through "WAIT_IIC_Code") to
;        ;read one byte of data from slave1 – the data is placed in
;        ;'single_data'. This action takes 2 bytes in the command file.
;        ;The next block writes this byte to slave2 using the same option.
;        ;
;Option2_file:
;        DB      slave1_address OR iicreadmask       ;slave address + read bit
;        DB      singleD_                             ;read into 'single_data'
;
;        DB      slave2_address OR iicwritemask     ;slave address + write bit
;        DB      singleD_                           ;get data from 'single_data'
;
;        DB      iicend
;
;MORE_PROGRAM:
;        continue with program
;
;EXAMPLE Option 5 & 6 & 8
;++++++++++++++++++++++++
;Not much more understanding gained by an example – see option 4 for doing
;a call in conjunction with any other option.
;
;EXAMPLE using other data spaces
;********************************
;This program can be used such that the command file resides in the internal
;DATA space or the external DATA space.  Examples demonstrating the utility
;of this feature will be described below.
;
;DATA Command File Example
;+++++++++++++++++++++++++
;            .
;            .
;            .
;DSEG
;iic_data:      DS      8                                      ;IIC data file
;datafile:      DS      5                  ;define a space for the command file
;            .
;            .
;CSEG
;       ;
;       ;load the command file into the DATA space.
;       ;
```

```
;PROGRAM:
;       MOV     datafile,#(slave1_address OR iicreadmask)
;       MOV     datafile+1,#ioD_
;       MOV     datafile+2,#8
;       MOV     datafile+3,#iic_data
;       MOV     datafile+4,#iicend
;
;P10:
;        MOV     IIC_Command_File_adrs,#datafile ;load address of command file
;                                               ;(notice only 1 byte required)
;        CALL    WAIT_IIC_Data         ;call program to wait for IIC execution
;
;The IIC commands will be executed from the DATA space starting at 'datafile'.
;This may not seem too useful, but it can be a very handy mechanism for
;communicating to a slave which is used often.  For example, assume that a
;system uses a PCF8570 IIC RAM for parameter storage.  The data in the RAM
;is required often, but the data required may be at any address in the RAM,
;and may be variable in length.  Using the DATA space as the command file,
;the programmer could construct a simple and compact mechanism for handling
;this system requirement:
;            .
;            .
;            .
;DSEG
;iic_data:      DS      8                                     ;IIC data file
;datafile:      DS      8                ;define a space for the command file
;            .
;            .
;CSEG
;       ;
;       ;load the command file into the DATA space.  The fist block sets the
;       ;subaddress for subsequent access to the RAM; the second block reads
;       ;the RAM.
;       ;
;PROGRAM:
;       MOV     datafile,#PCF8570_address OR iicwritemask)
;       MOV     datafile+1,#immediate_
;       MOV     datafile+2,#0
;
;       MOV     datafile+3,#(PCF8570_address OR iicreadmask)
;       MOV     datafile+4,#ioD_
;       MOV     datafile+5,#8
;       MOV     datafile+6,#iic_data
;       MOV     datafile+7,#iicend
;
;P10:
;       CALL    SUB       ;1st call will read 8 bytes from PCF8570 starting at
;            .                ;location 0 – bytes will be read into 'iic_data'
;            .
;            .
;       MOV     datafile+2,#7            ;change starting address to read from
;       MOV     datafile+5,#3                  ;change number of bytes to read
;       CALL    SUB             ;this call to "SUB" will read 3 bytes from the
;            .                 ;PCF8570 starting at location 7 – bytes will be
;            .                                         ;read into 'iic_data'
;            .
;       MOV     datafile+2,@R0           ;change starting address to read from
;       MOV     datafile+5,R7                  ;change number of bytes to read
;       CALL    SUB          ;this call to "SUB" will read the number of bytes
;            .               ;specified in R7 from the PCF8570 starting at the
;            .                 ;location specified by the DATA byte pointed to
;            .                     ;by R0 – bytes will be read into 'iic_data'
;
;       MOV     datafile+2,#33           ;change starting address to read from
;       MOV     datafile+5,#1                  ;change number of bytes to read
;       MOV     datafile+6,#B    ;change target internal address to register B
```

```
;       CALL    SUB              ;this call to ”SUB” will read 1 byte from the
;            .                ;PCF8570 starting at location 33 – the byte will
;            .                                        ;be placed in register B
;            .
;SUB:
;       MOV     IIC_Command_File_adrs,#datafile  ;load address of command file
;                                               ;(notice only 1 byte required)
;       CALL    WAIT_IIC_Data          ;call program to wait for IIC execution
;       RET
;
;One can construct a command file such that the called subroutine actually
;modifies the command file itself!  Also, the called subroutine could modify
;the ’IIC_Command_File_adrs’ register so that upon returning from the
;subroutine, a different IIC command file is executed other than the one
;immediately after the block that called the subroutine.
;
;XDATA Command File Example
;++++++++++++++++++++++++++
;            .
;            .
;            .
;DSEG
;iic_data:      DS      8                                       ;IIC data file
;XSEG
;datafile:      DS      5                  ;define a space for the command file
;            .
;            .
;CSEG
;       ;
;       ;load the command file into the XDATA space.
;       ;
;PROGRAM:
;       MOV     DPTR,#datafile
;       MOV     A,#(slave1_address OR iicreadmask)
;       MOVX    @DPTR,A
;       INC     DPTR
;       MOV     A,#ioD_
;       MOVX    @DPTR,A
;       INC     DPTR
;       MOV     A,#8
;       MOVX    @DPTR,A
;       INC     DPTR
;       MOV     A,#iic_data
;       MOVX    @DPTR,A
;       INC     DPTR
;       MOV     A,#iicend
;       MOVX    @DPTR,A
;
;P10:
;       MOV     IIC_Command_File_adrs,#LOW(datafile)          ;load address of
;       MOV     IIC_Command_File_adrs+1,#HIGH(datafile)          ;command file
;       CALL    WAIT_IIC_Xdata         ;call program to wait for IIC execution
;            .
;            .
;This program will run the command file from external data memory.  All the
;same options exist with XDATA command files as do with DATA command files.
;-----------------------------------------------------------------------------
;SYSTEM REQUIREMENTS:
;*******************
;Data Bytes Used:  15
;Bit Addressable Bytes Used: 1
;Bits used:  2
;Stack Penetration:  Approximately 17 bytes worst case
;Code Length:  Approximately 600 bytes of code.
;              NOTE: most of the code length of this program is made up of
;                    code used to take care of every generic addressing
;                    possibility.  If the user simply wants to use one known
```

```
;                    address space to hold the command file, and one target
;                    I/O address in a specific data space, the code and
;                    data requirements would be reduced dramatically.  This
;                    code is a good starting point for custom development
;                    since it is completely generic.
;
;"IIC_OS" register definitions:
;******************************
;The following data bytes are required to run the full implementation of the
;IIC driver system.  In a microcontroller as large as the 80C552 or
;80C652, the requirement of these data bytes will not impose a great toll
;on the user's system.  Note that registers, bytes, equates and bit names are
;surrounded by ' ' in the description – routines, subroutines and procedure
;names are surrounded by " ".
;
;------------
;'IIC_status'
;
;LOADED BY: "WAIT_IIC" and "IIC_VECTOR"
;DESCRIPTION – holds the status of the requested IIC operations.  This data
;byte is loaded with 'status_DO_IIC' by "WAIT_IIC", which then
;monitors this byte, and determines if the IIC command file has been
;completed.  Completion of the IIC command file is known if the 'IIC_status'
;is equal to one of the following:
;
;  'status_OK'                – operation complete, no problems
;  'status_arb_lost'          – arbitration lost to another master
;  'status_attempt_data'      – tried to send data 'max_data_attempts' times
;  'status_attempt_adrs'      – tried to find slave 'max_adrs_attempt' times
;  'status_timeout'           – waited 'max_wait' time for activity
;  'status_buss_err'          – a buss error (illegal start/stop)
;  'status_slave'             – addressed as slave (own address)
;  'status_arb_lost_slave'    – arbitration lost to another master, this one
;                               addressed as a slave
;  'status_general_slave'     – addressed as slave (general call)
;  'status_arb_lost_general'  – arbitration lost to a general call
;
;The values 'max_data_attempts', 'max_adrs_attempts' and 'max_wait' are
;equated in the main body below – these numbers define how many attempts
;should be made to send/receive data, locate a slave or wait for a response.
;
;----------------
;'IO_buffer_adrs'
;
;LOADED BY: "IIC_OS"
;DESCRIPTION:  is loaded by the "IIC_OS" operation and holds the address
;of the data to be transmitted to a slave, or received from a slave.  The bit–
;addressable register 'Command?adrs?space' determines which memory space the
;'IO_buffer_adrs' targets.  The initial value for this register can come from
;the command file itself, or may be loaded from the 'indirect_adrs' register,
;depending on the actions directed from the 'Command?adrs?space' register.
;
;-----------------------
;'IIC_Command_File_adrs'
;
;LOADED BY: calling routine, manipulated by "IIC_OS"
;DESCRIPTION: the calling routine loads the address of the command file to be
;executed by the "IIC_OS" into this two byte register.  The "IIC_OS" will
;modify this address register as the IIC_OS operations proceed. If the command
;file resides in the DATA space, then only the LSByte of the address is used.
;
;----------------
;'indirect_adrs'
;'indirect_count'
;
```

```
;LOADED BY: calling routine or a command file called subroutine
;DESCRIPTION: the calling routine may use the 'indirect_' option as
;loaded in the 'Command?adrs?space' byte.  This option directs the "IIC_OS"
;to get the 'IO_buffer_adrs' and the 'Multiple_count' from the calling routine
;(or called subroutine) loaded 'indirect_adrs' and 'indirect_count' registers.
;
;-------------
;'single_data'
;
;LOADED BY: calling routine or "IIC_VECTOR"
;DESCRIPTION: this byte allows for a quick mechanism to input one single byte
;of data from the IIC bus into the DATA space, or send 1 byte of data from the
;DATA space to the IIC bus.  If the IIC command file has the control code
;'singleD_', then the bit 'singleDATA' will be set in 'Command?adrs?space'.
;If 'singleDATA' is set, then the system will either input one byte of data
;only into 'single_data' or output the contents of 'single_data'.
;
;----------------
;'Multiple_count'
;
;LOADED BY: "IIC_OS"
;DESCRIPTION: this register is a counter for the number of bytes to be
;received or transmitted. Received or transmitted bytes are sent to or read
;from the address space indicated in 'Command?adrs?space' and addressed by
;the 'IO_buffer_adrs'.  The initial value for this register can come from
;the command file itself, or may be loaded from the 'indirect_count' register,
;depending on the actions directed from the 'Command?adrs?space' register.
;
;---------------
;'Attempt_count'
;
;LOADED BY: "IIC_OS"
;DESCRIPTION: counts the number of failed attempts at sending/receiving data
;or addressing a slave.  If the number of tries in either case exceeds
;'max_adrs_attempts' or 'max_data_attempts', the error status is reflected in
;'IIC_status' and the "IIC_OS" quits.
;
;-----------
;'last_data'
;
;LOADED BY: "IIC_OS"
;DESCRIPTION: holds the value of the last data byte received or transmitted.
;Used in "IIC_OS" as a look-back register so failed transmissions can be
;repeated.
;
;-----------
;'iic_timer'
;
;LOADED BY: "IIC_OS"
;DESCRIPTION: used as a watchdog timer for the IIC operation.  Implemented as
;a up-counter in "WAIT_IIC", but ideally, this function should be in the
;hands of a real system timer.
;
;----------------------
;'Slave_in & Slave_out'
;
;LOADED BY: mainline routine
;DESCRIPTION: Buffers for Slave receiver and Transmitter modes.  Main routine
;loads 'Slave_out' with the SLVbytes_out' number of bytes to be transmitted
;once addressed by another master.  'Slave_in' is filled by the 'SLVbytes_in'
;number of bytes from another master.  If more or less bytes are to be received
;or sent when addressed as a slave, then the size of the buffers and the
;'SLVbytes_... EQUates must change.
;
;IIC_OS does not need these 16 DATA bytes if the system is single-master.
;IF your system is single-master, then use the 'Slave_in' and 'Slave_out'
;buffers for your own general purpose buffers or registers.
```

Multimaster I^{2}C routines for byte-oriented I^{2}C interfaces AN435

```
;
;------
;'temp'
;
;This byte is used in the "WAIT_IIC" routine to hold the appropriate data
;space temporarily until the IIC bus is free or a slave receiver or slave
;transmitter mode is done.
;
;
;Slave address for this microcontroller - other bus masters can address this
;micro as a slave; this driver simply sends 'SLVbytes_out' number of bytes or
;receives 'SLVbytes_in' number of bytes in the case of being addressed as a
;slave.  The LSBit of the address is set indicating that general calls will
;be responded to.
;
Own_adrs        EQU     02EH       ;address of micro when addressed as a slave
general_enable  EQU     1          ;general call recognized since LSBit is set
SLVbytes_in     EQU     8        ;# bytes to receive when addressed as a slave
SLVbytes_out    EQU     8            ;# bytes to transmit when    "     "    "

DSEG    AT      48      ;change location to suit your system

IIC_status:             DS      1
IO_buffer_adrs:         DS      2
IIC_Command_File_adrs:  DS      2
indirect_adrs:          DS      2
indirect_count:         DS      1
iic_timer:              DS      2
Attempt_count:          DS      1
Multiple_count:         DS      1
last_data:              DS      1
single_data:            DS      1
temp:                   DS      1

Slave_in:               DS      SLVbytes_in
Slave_out:              DS      SLVbytes_out

DATA_start      EQU     Slave_out+Slvbytes_out
;
;'Command?adrs?space' is loaded by the calling routine, and manipulated by the
;"IIC_OS" routine.  It indicates which memory space the command file is to
;be read from.  It also ultimately gets the address space for the input and
;output data, as well as the indication for the special functions 'indirect_',
;'immediate_' and 'call_'.  Generally speaking, the calling routine loads
;'Command?adrs?space' with the code for which memory space holds the command
;file to be executed - then, the command file information amends this byte
;as each block of the command file is executed.
;
DSEG    AT      32     ;change location to suit your system - bit addressable!

Command?adrs?space:     DS      1

BSEG    AT      0                  ;address to match 'Command?adrs?space' byte

command_Data:           DBIT    1
command_Code:           DBIT    1
IO_Data:                DBIT    1
IO_Code:                DBIT    1
immediate_data:         DBIT    1
call_function:          DBIT    1
indirect_xxx:           DBIT    1
singleDATA:             DBIT    1
```

```
BSEG
;
;this next bit is set whenever the microcontroller is addressed as a slave
;receiver or a slave transmitter.  ”WAIT_IIC” needs this bit to hold off
;pending calls from the main routines to the IIC bus.
;
i_am_a_slave:           DBIT    1
;
;if any iic session failure occurs, set the failure bit
;
IIC_failure:            DBIT    1

BITS_start      EQU     IIC_failure+1
;
;mask bytes for the various ’Command?adrs?space’ bits.  These codes are used
;in the command file.  See examples above.
;
commandD_       EQU     00000001B               ;commands come from DATA space
commandC_       EQU     00000010B               ;commands come from CODE space
commandX_       EQU     00000000B              ;commands come from XDATA space
ioD_            EQU     00000100B           ;input/output data from DATA space
ioC_            EQU     00001000B                 ;output data from CODE space
ioX_            EQU     00000000B          ;input/output data from XDATA space
immediate_      EQU     00010000B                ;next byte is data to be sent
call_           EQU     00100000B                ;call subroutine after in/out
indirect_       EQU     01000000B          ;get info from ’indirect_ registers
singleD_        EQU     10000000B        ;get/put 1 byte to/from ’single_data’
commands_       EQU     00000011B           ;mask to isolate command adrs bits
iospaces_       EQU     00001100B              ;mask to isolate I/O space bits
specials_       EQU     11110000B                ;mask to isolate control bits
;
;The following status bytes are used to indicate the present state as well
;as the ultimate state of the IIC operations.  These values are loaded into
;’IIC_status’ by the various states as well as ”WAIT_IIC”.
;
status_OK               EQU     0          ;end of session and/or bus available
status_arb_lost         EQU     1                             ;arbitration lost
status_attempt_data     EQU     2 ;’max_data_attempts’ failed to get/send data
status_attempt_adrs     EQU     3    ;’max_adrs_attempts’ failed to find slave
status_timeout          EQU     4          ;”WAIT_IIC” detected timeout problem
status_buss_err         EQU     5            ;a bus error or illegal start/stop
status_slave            EQU     6             ;addressed as slave (own address)
status_general_slave    EQU     7            ;addressed as slave (general call)
status_arb_lost_slave   EQU     8         ;arbitration lost, addressed as slave
status_arb_lost_general EQU     9               ;arbitration lost, general call
status_DO_IIC           EQU     0FFH   ;all running fine (bus busy) indication
;
;IIC control characters.
;These characters are used in the IIC command file and various routines.
;’iicend’ must be the last character in any command file sequence.
;’iicwritemask’ and ’iicreadmask’ are used in conjunction with the slave
;                    address to indicate whether data is a comin’ or a goin’.
;’max_data_attempts’ should be equated to a value indicating how many times
;                    this system should re–try to get data from/to a slave
;                    before it gives up and says an error has occurred.
;’max_adrs_attempts’ should be equated to a value indicating how many times
;                    this system should re–try to address a slave
;                    before it gives up and says an error has occurred.
;’max_wait’ is used in a crude loop counting timer in ”WAIT_IIC”.  This number
;           should be equated to give roughly the timeout time required by
;           your system (i.e. IIC inactivity timeout).  The count will depend
;           on the clock speed as well as the average loop time in ”WAIT_IIC”.
;           The loop time will be affected by the IIC interrupt processing as
;           well as any other interrupt service routines in your system.
;
```

```
iicend                  EQU             0FFH          ;end of IIC command file
iicwritemask            EQU             00H
iicreadmask             EQU             01H
max_data_attempts       EQU             3       ;maximum tries to get/send data
max_adrs_attempts       EQU             3       ;maximum tries to address slave
max_wait                EQU             1         ;reload value for 'iic_timer'
                                                                  ;(counts up)
;
;'S1STA' reload values.  Virtually the same as old '552 app. note.
;
ENS1_NOTSTA_STO_NOTSI_NOTAA_CR0              EQU        0D1H
ENS1_NOTSTA_STO_NOTSI_AA_CR0                 EQU        0D5H
ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0              EQU        0C5H
ENS1_NOTSTA_NOTSTO_NOTSI_NOTAA_CR0           EQU        0C1H
ENS1_STA_NOTSTO_NOTSI_AA_CR0                 EQU        0E5H

;
;IIC hardware interrupt vector definition.

CSEG    AT      002BH
        JMP     IIC_VECTOR                   ;IIC_OS interrupt service routine
;
CSEG
;
;=============================================================================
;"WAIT_IIC" ;The calling routine has loaded the 'IIC_Command_File_adrs'
;register with the address of the command file to be executed.
;This routine simply waits for the IIC process to be completed.  Completion
;of the IIC session is indicated by 'IIC_status'= 'status_OK', or one of the
;many error codes. "IIC_VECTOR" will take care of updating the status byte.
;
;The calling routine must call the appropriate section of "WAIT_IIC" depending
;on which memory space the command file resides.  For example, if the command
;file resides in CODE memory space, the "WAIT_IIC_Code" must be called.
;
;It is possible that another master has addressed this master as a slave. If
;"WAIT_IIC" is called under these circumstances, it will exit with
;'IIC_failure' set and also check for a timeout for the addressed slave
;session.  If a timeout is detected, all IIC_OS registers and bits are set to
;their default value.  In either case the 'IIC_failure' bit is set.
;
;In the addressed slave mode, the calling master determines bus timing and
;clock values etc.  If something happens to that master, and it cannot
;complete it's session, then this slave is left hanging!  For this reason,
;we have a built-in timeout check feature for addressed slave mode (if and
;when this slave calls "WAIT_IIC" to do it's own work).
;
;INPUT:
;======
;'IIC_status' is checked to ensure an addressed slave state is not in progress
;OUTPUT:
;=======
;'IIC_status' is updated to reflect completion of IIC session or error.
;'IIC_failure' is updated (0 = all OK, 1 = some kind of error).
;=============================================================================
;
WAIT_IIC_Data:
        MOV     temp,#commandD_
        SJMP    WAIT_IIC
WAIT_IIC_Xdata:
        MOV     temp,#commandX_
        SJMP    WAIT_IIC
WAIT_IIC_Code:
        MOV     temp,#commandC_
WAIT_IIC:
        JNB      i_am_a_slave,WIIC_10
        ;
```

```
;if a slave receive or transmit session is in effect (as initiated by
;another master), this processor will only know that through the
;"IIC_VECTOR" routine – there is no timeout check etc.  Because of
:this situation, "WAIT_IIC" will check for a slave session in
;progress and exit as a failure if all is OK with that session (so
;that the calling routine will keep trying to get hold of the bus).
;If the slave session has timed out or there is an error, clear
;everything and exit as an error as well.
;
WIIC_05:
        CALL    IIC_time                        ;addressed slave timeout check
        JNZ     WIIC_35                              ;if not, exit as a failure
        WII_06: CLR i_am_a_slave             ;if timeout, reset system and exit
        SJMP    WIIC_ERR                                          ;as a failure
;
;ready to try – this micro is not presently addressed as a slave, and
;all else seems to be OK.
WIIC_10:
        MOV     Command?adrs?space,temp
        MOV     IIC_status,#status_DO_IIC                    ;indicate IIC busy
        SETB    STA               ;do an IIC interrupt (start start condition)
WIIC_15:
        MOV     iic_timer,#LOW(max_wait)                   ;start timeout timer
        MOV     iic_timer + 1,#HIGH(max_wait)
WIIC_20:
        CALL    IIC_time
        JZ     WIIC_EER

        ;
        ;as long as 'iic_timer' is OK, loop here and check the 'IIC_status'.
        ;'IIC_status' will remain as 'status_DO_IIC' as long as the IIC
        ;session is still on.  This byte will be loaded with 'status_OK' if
        ;the session ends normally, or it will be loaded with some other
        ;status byte if an error or arbitration process occurs.
        ;
        ;If this micro has been addressed as a slave, or has lost arbitration
        ;and become a slave, then 'IIC_status' indicates the situation, and
        ;this subroutine is terminated.  The routine that calls this one must
        ;check the 'IIC_status' to determine if another master has won the bus
        ;so that it can wait for 'IIC_status' to become 'status_OK', at which
        ;point, it could try again.
        ;
WIIC_25:
        MOV     A,IIC_status                    ;when = status_OK, all done
        CJNE    A,#status_DO_IIC,WIIC_30
        SJMP     WIIC_20
WIIC_30:
        CJNE    A,#status_OK,WIIC_35
WIIC_X:
        CLR     IIC_failure
        CLR     i_am_a_slave
        RET
        ;
        ;if 'iic_timer' overflows, have an IIC bus timeout error.
        ;
WIIC_ERR:
        CALL    MORE_00_SUB                                ;clear all registers
        ANL     IIC_status, #11110000B
        ORL     IIC_status,#status_timeout                 ;indicate inactivity
        CALL    WIIC_ERRX
        ;
        ;do error recovery here – i.e. lost arbitration, bus error.
        ;Alternately, can return the error code to the calling routine so
        ;that the main routines decide what to do for various errors.  For
        ;development and debug purposes, this example routine ignores the
        ;errors.
        ;
```

```
WIIC_35:
        SETB    IIC_failure                     ;indicate a failed IIC session
        RET                               ;ensure the interrupt bit is cleared
WIIC_ERRX:
        RETI
        ;
;======
;"IIC_time"
        ;Subroutine to increment IIC timeout timer.  ACC returns 0 if timeout
        ;occurs.
        ;
IIC_time:
        INC     iic_timer
        MOV     A,iic_timer
        JNZ     time_X
        INC     iic_timer + 1
        MOV     A,iic_timer + 1
time_X:
        RET
;
;============================================================================
;Subroutine "FETCH_DATA"
;DESCRIPTION:
;============
;This subroutine is used by all the IIC_OS states to get the next data byte
;from the address 'IO_address' in the memory space indicated in
;'Command?adrs?space'.  This routine also saves the fetched data in the byte
;'last_data' so error recovery can be easily done.  Before this routine exits,
;the pointer 'IO_buffer_adrs' is incremented.
;Fetched data returned in ACCumulator.
;INPUT:
;======
;'IO_address' has address where data is to be fetched from.  If the target
;             space is DATA, then the LSByte of 'IO_address' is the full
;             address and the MSByte is ignored.
;'Command?adrs?space' holds the information for which address space is to be
;             targeted for fetching the data.
;OUTPUT:
;=======
;ACCumulator is loaded with the fetched byte.
;'last_data' gets the fetched byte as well.
;'IO_address' is incremented.
;
FETCH_DATA:
        JB      IO_Data,FD_Data                       ;is data in DATA space?
        MOV     DPL,IO_buffer_adrs       ;no, then must be XDATA or CODE space
        MOV     DPH,IO_buffer_adrs+1
        JB      IO_Code,FD_Code                         ;is it in CODE space?

FD_Xdata:
        MOVX    A,@DPTR                     ;no, it must be in XDATA space

FD_exit:
        MOV     last_data,A                                      ;store data
        INC     DPTR                                           ;bump pointer
        MOV     IO_buffer_adrs,DPL                          ;restore pointer
        MOV     IO_buffer_adrs + 1,DPH
        RET
        ;
        ;enter here if data is in CODE space
        ;
FD_Code:
        CLR     A
        MOVC    A,@A+DPTR
        SJMP    FD_exit
        ;
        ;enter here if data is in DATA space
```

```
        ;
FD_Data:
        MOV     R0,IO_buffer_adrs
        MOV     A,@R0
        MOV     last_data,A
        INC     R0
        MOV     IO_buffer_adrs,R0
        RET
;
;===============================================================================
;Subroutine "STORE_DATA"
;DESCRIPTION:
;============
;This subroutine stores incoming data in the address 'IO_address' in the
;data space indicated in 'Command?adrs?space'.  Only XDATA and DATA spaces
;are valid since we cannot write into the CODE space.
;INPUT:
;======
;ACCumulator has data to be stored.  This data is not corrupted.
;'IO_address' holds address for where data is to be stored
;'Command?adrs?space' (bit addressable) describes which data space is to
;             be targeted.
;OUTPUT:
;=======
;ACCumulator contents not corrupted by subroutine
;ACCumulator contents are stored in address as described above.
;'last_data' holds a copy of the data.
;'IO_address' is incremented.
;
STORE_DATA:
        JB      IO_Data,SD_Data                          ;is target area DATA?

SD_Xdata:
        MOV     DPL,IO_buffer_adrs             ;no, then must be XDATA so load
        MOV     DPH,IO_buffer_adrs+1                        ;address into DPTR
        MOVX    @DPTR,A                                        ;store the data
        MOV     last_data,A                                       ;and copy it
        INC     DPTR                                 ;bump the address pointer
        MOV     IO_buffer_adrs,DPL                             ;and restore it
        MOV     IO_buffer_adrs+1,DPH
        RET
        ;
        ;enter here if the target space is DATA
        ;
SD_Data:
        MOV     R0,IO_buffer_adrs                     ;get the address into R0
        MOV     @R0,A                                          ;store the data
        MOV     last_data,A                                       ;and copy it
        INC     R0                                   ;bump the address pointer
        MOV     IO_buffer_adrs,R0                              ;and restore it
        RET
;
;===============================================================================
;Subroutine "FETCH_COMMAND"
;DESCRIPTION:
;============
;This subroutine fetches a byte from the address 'IIC_Command_File_adrs' in
;the address space indicated in 'Command?adrs?space' (bit addressable).  If
;"FETCH_COMMAND_0" is called, then the address pointer 'IIC_Command_File_adrs'
;is not incremented at exit, otherwise the address pointer is incremented.
;INPUT:
;======
;'IIC_Command_File_adrs' holds the address of the byte to be retrieved.
;'Command?adrs?space' indicates in which address space the command file resides
```

```
;OUTPUT:
;=======
;ACCumulator holds the retrieved byte.
;'IIC_Command_File_adrs' is incremented (not if "FETCH_COMMAND_0" is called).
;
FETCH_COMMAND_0:
        CLR     C                  ;carry indicates whether pointer inc or not
        SJMP    FC_10

FETCH_COMMAND:
        SETB    C

FC_10:
        JB      command_Data,FC_Data           ;is command file in DATA space?
        MOV     DPL,IIC_Command_File_adrs           ;no, must be XDATA or CODE
        MOV     DPH,IIC_Command_File_adrs+1
        JB      command_Code,FC_Code           ;is command file in CODE space?

FC_Xdata:
        MOVX    A,@DPTR                             ;no, then must be in XDATA

FC_exit:
        JNC     FCX_10                    ;don't increment pointer if no carry
        INC     DPTR                          ;if carry set, increment pointer
FCX_10:
        MOV     IIC_Command_File_adrs,DPL                      ;restore pointer
        MOV     IIC_Command_File_adrs+1,DPH
        RET
        ;
        ;enter here to retrieve command byte from CODE space
        ;
FC_Code:
        CLR     A
        MOVC    A,@A+DPTR
        SJMP    FC_exit
        ;
        ;enter here to retrieve command byte from DATA space
        ;
FC_Data:
        MOV     R0,IIC_Command_File_adrs
        MOV     A,@R0
        JNC     FCD_X
        INC     R0
FCD_X:
        MOV     IIC_Command_File_adrs,R0
        RET
;
;=============================================================================
;Subroutine "make_space"
;DESCRIPTION:
;============
;"make_space" is used to update the 'Command?adrs?space' byte which holds
;the information for which memory space is to be targeted for data and
;command bytes.  This subroutine is usually called by a state that has just
;read-in the command file byte indicating address space information.
;INPUT:
;======
;ACCumulator has the data address space indication read-in from command file.
;OUTPUT:
;=======
;'Command?adrs?space' is updated with ACCumulator contents.
;
make_space:
        XCH     A,Command?adrs?space           ;get present address space byte
        ANL     A,#commands_               ;clear all bits except command bits
        ORL     A,Command?adrs?space           ;mask in new address space info
        XCH     A,Command?adrs?space                                  ;restore
```

```
ms_X:
        RET

$eject
;
;==============================================================================
;IIC interrupt vector
;Every time a significant event occurs on the IIC bus (a start, stop, error,
;etc.), this interrupt routine is entered.  This routine reads the IIC
;hardware SFR called 'S1STA' to determine what state the IIC hardware is in.
;Each state has it's own processing routine as shown below.
;The multimaster routines shown are very simple in this module - multimaster
;functions are very dependent on the system being serviced.  This
;module simply relinquishes control of the bus if another master wins
;arbitration;  it will receive or send bytes if it is addressed as a
;slave.
;==============================================================================
;
IIC_VECTOR:
        PUSH    PSW               ;save all registers used in interrupt vector
        PUSH    ACC
        PUSH    DPL
        PUSH    DPH
        PUSH    AR0
;
;'S1STA', the SFR indicating IIC hardware status for the '552 takes on a
;limited range of values, namely 00H to 0C8H in steps of 08H.  The following
;manipulation changes the 'S1STA' value to a number from 0 to 25.  This
;number is then multiplied by 2 so a jump can be done from an 'AJMP' table.
;
        MOV     A,S1STA            ;get SFR which holds hardware status of bus
        SWAP    A
        RLC     A
        JNC     IICV_10
        INC     A
IICV_10:
        RL      A
        MOV     DPTR,#S1STA_00
        JMP     @A+DPTR
;
;all sections exit here.
;The timeout timer 'iic_timer' is restarted every time around, it is assumed
;that if an interrupt occurs, that more than likely, everything is OK.
;
IIC_EXIT:
        MOV     iic_timer,#LOW(max_wait)                 ;reload timeout timer
        MOV     iic_timer + 1,#HIGH(max_wait)

        POP     AR0
        POP     DPH
        POP     DPL
        POP     ACC
        POP     PSW
        RETI
;
;------------------------------------------------------------------------------
;Jump table for interrupt routine entry above.
;------------------------------------------------------------------------------
;
S1STA_00:
        AJMP     MORE_00                                       ;Bus Error mode

        AJMP     MORE_08                    ;Master Receiver/Transmitter Mode
        AJMP     MORE_10                    ;Master Receiver/Transmitter Mode
        AJMP     MORE_18                             ;Master Transmitter Mode
        AJMP     MORE_20                             ;Master Transmitter Mode
        AJMP     MORE_28                             ;Master Transmitter Mode
```

```
        AJMP     MORE_30                                  ;Master Transmitter Mode
        AJMP     MORE_38                         ;Master Receiver/Transmitter Mode

        AJMP     MORE_40                                     ;Master Receiver Mode
        AJMP     MORE_48                                     ;Master Receiver Mode
        AJMP     MORE_50                                     ;Master Receiver Mode
        AJMP     MORE_58                                     ;Master Receiver Mode

        AJMP     MORE_60                                      ;Slave Receiver Mode
        AJMP     MORE_68                                      ;Slave Receiver Mode
        AJMP     MORE_70                                      ;Slave Receiver Mode
        AJMP     MORE_78                                      ;Slave Receiver Mode
        AJMP     MORE_80                                      ;Slave Receiver Mode
        AJMP     MORE_88                                      ;Slave Receiver Mode
        AJMP     MORE_90                                      ;Slave Receiver Mode
        AJMP     MORE_98                                      ;Slave Receiver Mode
        AJMP     MORE_A0                                      ;Slave Receiver Mode

        AJMP     MORE_A8                                   ;Slave Transmitter Mode
        AJMP     MORE_B0                                   ;Slave Transmitter Mode
        AJMP     MORE_B8                                   ;Slave Transmitter Mode
        AJMP     MORE_C0                                   ;Slave Transmitter Mode
        AJMP     MORE_C8                                   ;Slave Transmitter Mode
;
;-----------------------------------------------------------------------------
;State 00 = Bus error due to an illegal START or STOP condition.  This state
;can also occur if the SIO1 enters an undefined state.
;-----------------------------------------------------------------------------
;
MORE_00:
        MOV     IIC_status,#status_buss_err                 ;indicate bus error
        ANL     P1,#00111111B                                       ;unstick bus
        ORL     P1,#11000000B
M00_10:
        CALL    MORE_00_SUB          ;clear all "IIC_OS" status counters etc.
        JMP     IIC_EXIT
;
;This portion of State 00 was made into a subroutine so that the "WAIT_IIC"
;routine could call it when a timeout error occurs.
;This routine sets all counters and other "IIC_OS" registers to 0.
;
MORE_00_SUB:
        CLR     i_am_a_slave
        CLR     A
        MOV     Attempt_count,A
        MOV     Multiple_count,A
        MOV     last_data,A
        MOV     S1CON,#ENS1_NOTSTA_STO_NOTSI_AA_CR0                       ;STOP
        RET
;
;-----------------------------------------------------------------------------
;State 08 indicates that a start condition has been transmitted.
;MASTER RECEIVER/TRANSMITTER MODE.
;In this case, the "IIC_OS" possibilities are one - the next byte in the
;'IIC_Command_FIle' must be the slave address (and read/write bit).
;-----------------------------------------------------------------------------
;
MORE_08:
        CALL    FETCH_COMMAND                   ;get next byte in command file
        MOV     S1DAT,A                                             ;transmit it
M08_10:
        MOV     S1CON,#ENS1_NOTSTA_NOTSTO_NOTSI_NOTAA_CR0
        JMP     IIC_EXIT
;
```

Multimaster I²C routines for byte-oriented I²C interfaces AN435

```
;-----------------------------------------------------------------------------
;State 10H = a repeated start condition has been transmitted.
;MASTER RECEIVER/TRANSMITTER MODE.
;This state is handled just like State 08H.  The ”IIC_OS” definition ensures
;that the next byte in the ’IIC_Command_File’ will be a slave address (and
;read/write bit).
;-----------------------------------------------------------------------------
;
MORE_10:
        JMP     MORE_08
;
;-----------------------------------------------------------------------------
;State 18H = (Slave address + Write bit) has been transmitted, and an ACK has
;             been returned.
;MASTER TRANSMITTER MODE.
;Once a slave address has been transmitted, several possibilities exist,
;namely: set–up to send/receive n bytes with count and address info coming
;         from the command file stream
;                                      OR
;         set–up to send/receive n bytes with count and address info coming
;         from the ’indirect_count’ and ’indirect_adrs’ registers.  The
;         mainline routine must set these registers before initiating an IIC
;         session.
;                                      OR
;         set–up to send 1 byte (’immediate_’) from the command file stream.
;                                      OR
;         set–up to send/receive 1 byte from/to ’single_data’ register.  The
;         mainline routine must load ’single_data’ if it is to be transmitted.
;-----------------------------------------------------------------------------
;
MORE_18:
        MOV     Attempt_count,#0                   ;clear failed attempt count
M18_15:
        CALL    FETCH_COMMAND                   ;get next byte in command file
        CALL    make_space                        ;update ’Command?adrs?space’
        JBC     immediate_data,M18_30                       ;is data immediate?
        CALL    M18_SUB                     ;No, call subroutine to load count
M18_20:                                                    ;and address of data
        CALL    FETCH_DATA                         ;now ready to get data byte
M18_25:
        MOV     S1DAT,A                                           ;send as data
M18_X:
        MOV     S1CON,#ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0
        JMP     IIC_EXIT
        ;
        ;enter here if immediate data output requested.
        ;The data to be transmitted is the next byte in the command file.
        ;
M18_30:
        MOV     Multiple_count,#0
        CALL    FETCH_COMMAND
        SJMP    M18_25
        ;
        ;”M18_SUB” subroutine checks for ’indirect_’ and ’singleD_’ commands.
        ;State 18H and State 40H use this subroutine.
        ;If an indirect feature is requested, load address and count
        ;information from the ’indirect_count’ and ’indirect_adrs’ registers
        ;if the ’indirect_’ feature is not requested, then the count and
        ;address information are contained in the next bytes of the command
        ;file.
        ;
M18_SUB:
        JBC     indirect_xxx,M18S_10       ;if indirect, clear bit and service
        JBC     singleDATA,M18S_20      ;if one data byte in/out to ’iic_data’
        ;
        ;enter here if the count and address for the data to be read/written
        ;is in the command file itself (i.e. no special commands).
```

```
        ;
        CALL    FETCH_COMMAND                   ;get next byte in command file
        DEC     A                                               ;decrement and
        MOV     Multiple_count,A                          ;store as byte count
        CALL    FETCH_COMMAND                   ;get next byte in command file
        MOV     IO_buffer_adrs,A              ;store as LSByte of data address
        JB      IO_Data,M18S_X             ;if DATA space, only 1 address byte
        CALL    FETCH_COMMAND                   ;get next byte in command file
        MOV     IO_buffer_adrs+1,A            ;store as MSByte of data address
M18S_X:
        RET
        ;
        ;enter here if indirect requested.  The number of bytes to be
        ;written or read is contained in the 'indirect_count' register,
        ;the address of the bytes to be read or written is contained in
        ;the 'indirect_address' register(s).
        ;
M18S_10:
        DEC     indirect_count
        MOV     Multiple_count,indirect_count
        MOV     IO_buffer_adrs,indirect_adrs
        MOV     IO_buffer_adrs + 1,indirect_adrs + 1
        RET
        ;
        ;enter here if single byte input/output from DATA space requested
        ;('singleDATA').
        ;
M18S_20:
        MOV     Multiple_count,#0
        MOV     A,#NOT(iospaces_)
        ANL     A,Command?adrs?space
        ORL     A,#ioD_
        MOV     Command?adrs?space,A
        MOV     IO_buffer_adrs,#single_data
        RET
;
;----------------------------------------------------------------------------
;State 20H = (Slave address + Write bit) has been transmitted, no ACK from
;slave.
;MASTER TRANSMITTER MODE.
;This state counts the number of failures for a transmitted address - if
;'max_adrs_attempts' failures occur in-a-row, then abort session.
;----------------------------------------------------------------------------
;
MORE_20:
        INC     Attempt_count                              ;bump attempt count
        MOV     A,Attempt_count
        CJNE    A,#max_adrs_attempts,M20_10            ;if too many failures,
        MOV     IIC_status,#status_attempt_adrs       ;indicate attempt error
        JMP     M00_10
        ;
        ;if less than 'max_attempts' failures, then set command file pointer
        ;back one, and try sending address again.
        ;
M20_10:
        MOV     R0,#IIC_Command_File_adrs
        MOV     A,@R0
        JNZ     M20_20
        INC     R0
        DEC     @R0
        DEC     R0
M20_20:
        DEC     @R0

        JMP     M08_10
;
```

```
;---------------------------------------------------------------------------
;State 28H = Data byte has been transmitted, ACK has been received.
;MASTER TRANSMITTER MODE.
;This section is entered when a data byte has been successfully transmitted.
;Now the system has to check if more data bytes are to be sent, and if not,
;should a subroutine be called before going on to next IIC block in the
;IIC command file.
;---------------------------------------------------------------------------
;
MORE_28:
        MOV     Attempt_count,#0            ;clear attempt count since all OK
        MOV     A,Multiple_count             ;check for end of data bytes out
        JZ      M28_03                               ;last byte has been sent
        DEC     Multiple_count             ;more bytes to be sent so decrement
        JMP     M18_20                              ;count and send next byte
        ;
        ;enter here when all data bytes sent.
        ;
M28_03:
        JBC     call_function,M28_20     ;check for request to call subroutine
M28_05:
        CALL    FETCH_COMMAND_0                     ;check for end–of–session
        CJNE    A,#iicend,M28_10
M28_X:
        MOV     IIC_status,#status_OK
        JMP     M00_10
        ;
        ;if not end–of–session, do another start
        ;
M28_10:
        MOV     S1CON,#ENS1_STA_NOTSTO_NOTSI_AA_CR0
        JMP     IIC_EXIT
        ;
        ;enter here if a ’call_’ to a subroutine is requested.  First push
        ;the return address (above) onto stack, then get the address of the
        ;subroutine to call from the IIC command file.  Push the call address
        ;onto the stack (low address first), then call subroutine.
        ;
        ;The called subroutine could be used to modify the contents of the
        ;’IIC_Command_File_adrs’ registers.  In doing so, IF–THEN–ELSE
        ;control flow could be done (i.e. based on some  IIC read information,
        ;the subroutine may decide to run one of several other IIC blocks,
        ;or end the session altogether).  More likely, the subroutine will be
        ;used to manipulate some data before it is transmitted.
        ;
M28_20:
        MOV     A,#LOW(M28_05)                  ;put return address onto stack
        PUSH    ACC
        MOV     A,#HIGH(M28_05)
        PUSH    ACC
        CALL    FETCH_COMMAND     ;get address of subroutine from command file
        PUSH    ACC                     ;and put it onto the stack (LSB first)
        CALL    FETCH_COMMAND
        PUSH    ACC
        RET                                                ;CALL the subroutine
;
;---------------------------------------------------------------------------
;State 30H = Data byte has been transmitted, NO ACK received.
;MASTER TRANSMITTER MODE.
;This state is similar to state 20H, except that data has been transmitted,
;not an address.
;The routine ’FETCH_DATA’ always stores the data fetched as ’last_data’ so
;that in the case of a NO ACK, it can be re–transmitted.
;---------------------------------------------------------------------------
;
```

```
MORE_30:
        INC     Attempt_count                              ;bump attempt count
        MOV     A,Attempt_count                    ;if too many attempt, error
        CJNE    A,#max_data_attempts,M30_10
        MOV     IIC_status,#status_attempt_data          ;error status update
        JMP     M00_10

M30_10:
        MOV     A,last_data                         ;get data not received and
        JMP     M18_25                                            ;re-send it
;
;----------------------------------------------------------------------------
;State 38H = Arbitration lost to another master.
;MASTER TRANSMITTER MODE.
;If this state is entered, simply let the other Master have the run of the
;bus.  The mainline routine that started the IIC session can check
;the 'IIC_status' register for this state and re-try later.
;----------------------------------------------------------------------------
;
MORE_38:
        MOV     IIC_status,#status_arb_lost              ;error status update
        JMP     M40_20
;
;----------------------------------------------------------------------------
;State 40H = (Slave Address + Read bit) has been transmitted, ACK received.
;MASTER RECEIVER MODE.
;This state is very similar to State 18H and shares a subroutine with that
;state.
;----------------------------------------------------------------------------
;
MORE_40:
        MOV     Attempt_count,#0
M40_15:
        CALL    FETCH_COMMAND
        CALL    make_space
        CALL    M18_SUB
M40_19:
        MOV     A,Multiple_count
        JNZ     M40_20
        MOV     S1CON,#ENS1_NOTSTA_NOTSTO_NOTSI_NOTAA_CR0
        JMP     IIC_EXIT
M40_20:
        MOV     S1CON,#ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0
        JMP     IIC_EXIT
;
;----------------------------------------------------------------------------
;State 48H = (Slave address + Read bit) transmitted, NOT ACK received.
;MASTER RECEIVER MODE.
;See State 20H.
;----------------------------------------------------------------------------
;
MORE_48:
        JMP     MORE_20
;
;----------------------------------------------------------------------------
;State 50H = Data byte has been received, ACK returned.
;MASTER RECEIVER MODE.
;This state stores the received data byte and determines whether more data is
;required or not.  If more data required (i.e. 'Multiple_count' > 1), then
;send back an ACK, if no more data to be received ('Multiple_count" = 1), then
;set-up to return a NOT ACK on next data byte reception.
;----------------------------------------------------------------------------
;
MORE_50:
        MOV     A,S1DAT                                ;get received data byte
        CALL    STORE_DATA                ;store the data in appropriate space
        MOV     Attempt_count,#0                                ;indicate all OK
```

```
        MOV     A,Multiple_count          ;check for more bytes to be received
        CJNE    A,#1,M50_10
        ;
        ;If next byte to be received is last, make sure a NOT ACK is sent
        ;with next reception.
        ;
M50_05:
        MOV     S1CON,#ENS1_NOTSTA_NOTSTO_NOTSI_NOTAA_CR0
        SJMP    M50_15
M50_10:
        MOV     S1CON,#ENS1_NOTSTA_NOTSTO_NOTSI_AA_CR0
M50_15:
        DEC     Multiple_count
        JMP     IIC_EXIT
;
;---------------------------------------------------------------------------
;State 58H = Data byte has been received, NOT ACK has been returned.
;MASTER RECEIVER MODE.
;This state is entered when the last byte required has been received by the
;Master.  In this case, the byte must be stored, then a check must be done
;for the calling of a subroutine, and/or the end of the entire IIC session.
;See State 28H for more details.
;---------------------------------------------------------------------------
;
MORE_58:
        MOV     A,S1DAT                                   ;get received byte
        CALL    STORE_DATA                                         ;store it
        MOV     Attempt_count,#0                        ;clear error flag
        JMP     M28_03                    ;check for end–of–session or 'call_'
;
;---------------------------------------------------------------------------
;State 60H = Own Slave Address (+ Write bit) has been received,
;            ACK has been returned.
;SLAVE RECEIVER MODE.
;When own address found, this system will receive 'SLVbytes_in' bytes of data
;into 'Slave_in' data space.
;The calling master must produce the stop or repeated start conditions.  This
;micro was not in an active IIC mode when the other master addressed it, so
;the "WAIT_IIC" subroutine is not active, thus timeout problems will not be
;checked for unless "WAIT_IIC" is called.  "WAIT_IIC" will do only a timeout
;check if called from the main program since it will wait for the 'IIC_status'
;to become 'status_OK'.
;---------------------------------------------------------------------------
;
MORE_60:
        MOV     IIC_status,#status_slave
M60_10:
        SETB    i_am_a_slave
        MOV     Multiple_count,#(SLVbytes_in – 1)    ;set for # bytes received
        MOV     Command?adrs?space,#ioD_        ;receive bytes into DATA space
        MOV     IO_buffer_adrs,#Slave_in         ;address of DATA space target
        SJMP    M40_20
;
;---------------------------------------------------------------------------
;State 68H = Arbitration lost while addressing a slave; Own slave address and
;            write bit has been received.
;SLAVE RECEIVER MODE.
;Indicate that arbitration is lost so that the "WAIT_IIC" routine is aborted
;and the interrupt from the IIC hardware runs the system.
;"WAIT_IIC" is active if this state is entered since state 68H is entered
;upon lost arbitration for the bus.
;"WAIT_IIC" will terminate in this case since the 'IIC_status' will show that
;another master has won the bus.
;---------------------------------------------------------------------------
;
```

```
MORE_68:
        MOV     IIC_status,#status_arb_lost_slave
        SJMP    M60_10
;
;-----------------------------------------------------------------------------
;State 70H = General call address (00H) has been received, ACK has been
;            returned (by this micro).
;SLAVE RECEIVER MODE.
;Indicates that a general call has been received – 'SLVbytes_in' bytes will
;be received into 'Slave_in' as if this slave were addressed.
;-----------------------------------------------------------------------------
;
MORE_70:
        MOV     IIC_status,#status_general_slave
        SJMP    M60_10
;
;-----------------------------------------------------------------------------
;State 78H = Arbitration lost while addressing a slave – General call address
;            (00H) has been received, ACK has been returned (by this micro).
;SLAVE RECEIVER MODE.
;Indicates that a general call has been received – 'SLVbytes_in' bytes will be
;received into 'Slave_in' as if this slave were addressed.
;-----------------------------------------------------------------------------
;
MORE_78:
        MOV     IIC_status,#status_arb_lost_general
        SJMP    M60_10
;
;-----------------------------------------------------------------------------
;State 80H = Previously addressed with own slave address; data has been
;            received, ACK has been returned (by this micro).
;SLAVE RECEIVER MODE.
;Data byte received in 'S1DAT', ACK returned.
;-----------------------------------------------------------------------------
;
MORE_80:
        SJMP    MORE_50
;
;-----------------------------------------------------------------------------
;State 88H = Previously addressed with own slave address; data byte has been
;            received, NOT ACK has been returned (by this micro).
;SLAVE RECEIVER MODE.
;Last byte to be received is in 'S1DAT'.  A NACK has been returned.
;-----------------------------------------------------------------------------
;
MORE_88:
        MOV     A,S1DAT                                    ;get received byte
        CALL    STORE_DATA                                          ;store it
M88_10:
        MOV     IIC_status,#status_OK
        CLR     i_am_a_slave
        SJMP    M40_20
;
;-----------------------------------------------------------------------------
;State 90H = Previously addressed with general call; data byte has been
;            received, ACK has been returned (by this micro).
;SLAVE RECEIVER MODE.
;-----------------------------------------------------------------------------
;
MORE_90:
        SJMP    MORE_80
;
```

```
;-----------------------------------------------------------------------------
;State 98H = Previously addressed with general call; data byte has been
;            received, NOT ACK has been returned (by this micro).
;SLAVE RECEIVER MODE.
;-----------------------------------------------------------------------------
;
MORE_98:
        SJMP    MORE_88
;
;-----------------------------------------------------------------------------
;State A0H = a STOP or repeated START has been received while still in the
;            addressed slave receiver or transmitter mode.
;SLAVE RECEIVER MODE.
;-----------------------------------------------------------------------------
;
MORE_A0:
        SJMP    M88_10
;
;-----------------------------------------------------------------------------
;State A8H = Own slave address + read byte has been received; ACK has been
;            returned (by this micro).
;SLAVE TRANSMITTER MODE.
;This micro has been addressed by another master, and has been told to send
;data.  This micro will respond by sending 'SLVbytes_out' bytes of data from
;'Slave_out'.
;-----------------------------------------------------------------------------
;
MORE_A8:
        MOV     IIC_status,#status_slave
MA8_10:
        SETB    i_am_a_slave
        MOV     Multiple_count,#(SLVbytes_out)     ;set for 2 bytes to be sent
        MOV     Command?adrs?space,#ioD_       ;transmit bytes from DATA space
        MOV     IO_buffer_adrs,#Slave_out        ;address of DATA space target
        MOV     IO_buffer_adrs + 1,#0
        SJMP    MORE_B8
;
;-----------------------------------------------------------------------------
;State B0H = Arbitration lost while trying to get to a slave; own slave
;            address + read has been received; ACK has been returned (by this
;            micro).
;SLAVE TRANSMITTER MODE.
;-----------------------------------------------------------------------------
;
MORE_B0:
        MOV     IIC_status,#status_arb_lost_slave
        SJMP    MA8_10
;
;-----------------------------------------------------------------------------
;State B8H = Data byte in S1DAT has been transmitted; ACK has been received.
;SLAVE TRANSMITTER MODE.
;This section checks if any more bytes are to be transmitted.
;-----------------------------------------------------------------------------
;
MORE_B8:
        MOV     A,Multiple_count
        JZ      MB8_03
        DEC     Multiple_count
MB8_03:
        CALL    FETCH_DATA                         ;now ready to get data byte
        MOV     S1DAT,A                                          ;send as data
        JMP     M40_19
;
```

```
;-----------------------------------------------------------------------------
;State C0H = Data byte in S1DAT has been transmitted; NOT ACK has been
;            received.
;SLAVE TRANSMITTER MODE.
;This is the end of the addressed slave session.  A STOP or repeated START
;will be the next state, but this addressed slave doesn't care unless the next
;address sent by the calling master is it's own, or the general call address.
;-----------------------------------------------------------------------------
;
MORE_C0:
        JMP     M88_10
;
;-----------------------------------------------------------------------------
;State C8H = Last data byte in S1DAT has been transmitted; ACK has been
;            received.
;SLAVE TRANSMITTER MODE.
;Treated same as state C0.
;-----------------------------------------------------------------------------
;
MORE_C8:
        JMP     M88_10

CODE_start      EQU     $

}
```