Using the 8XC751/752 in multimaster I^{2}C applications　AN430

INTRODUCTION

The Philips Semiconductors 83C751/87C751 offers the advantages of the 80C51 architecture in a small package and at a low cost. It combines the benefits of a high performance microcontroller with on-board hardware supporting the Inter Integrated Circuit (I^{2}C) bus interface.

The Inter IC (I^{2}C) bus developed by Philips allows integrated circuits to communicate directly with each other via a simple bidirectional 2-wire bus. The comprehensive family of CMOS and bipolar ICs incorporating the on-chip I^{2}C interface offers many advantages to designers of digital control for industrial, consumer and telecommunications equipment.

Interfacing the devices in an I^{2}C based system is very simple as they connect directly to the two bus lines: a serial data line (SDA) and a serial clock line (SCL). System design can rapidly progress from block diagram to final schematics, as there is no need to design bus interfaces. In addition, functional blocks on the block diagram correspond to actual ICs. A prototype system or a final product version can be easily modified or upgraded by ‘clipping’ or ‘unclipping’ ICs to or from the bus. The simplicity of designing with the I^{2}C bus does not reduce its effectiveness: it is a reliable, multimaster bus with integrated addressing and data-transfer protocols. The I^{2}C-bus compatible ICs give cost reduction benefits through smaller IC packages and a minimization of PCB traces and glue logic.

The availability of microcontrollers, like the 83C751, with on-board I^{2}C interface is a very powerful tool for system designers. The integrated protocols allow systems to be completely software defined. Software development time of different products can be reduced by assembling a library of re-usable software modules. In addition, the multimaster capability allows rapid testing and alignment of end-products via external connections to an assembly-line computer.

The mask programmable 83C751 and its EPROM version, 87C751, can operate as a master or a slave device on the I^{2}C small area network. In addition to the efficient interface to the dedicated function ICs in the I^{2}C family the on-board interface facilitates I/O and RAM expansion, access to EEPROM, and processor-to-processor communications.

The 83C752 and its EPROM version, 87C752, are essentially the 83C751/87C751 with the addition of a five channel multiplexed 8-bit A/D converter and an 8-bit PWM output. As the I^{2}C bus interface is identical, the programming example and the discussion relates to both processors. The multimaster capability of the I^{2}C bus allows easy integration and expansion of relatively complex systems, in which different devices can independently initiate data transfers. Integration of a multimaster system is easy as a Master on the bus does not have to coordinate its data transfer with other potential Master devices—arbitration and synchronization are taken care of by the hardware and bus protocols. Expanding a system with a new device is trivial—it is “clipped” onto the two serial bus lines, and the new device may act as a Master without any modification to the other devices (see Figure 1). Microcontrollers like the S8XC751/752 on the I^{2}C bus are extremely powerful, as they can be programmed to be both Masters and Slaves in the same system. This way the microcontroller may initiate communication on the bus, and when requested, will respond to a data transfer request by another device.

In this Application Note we shall discuss the most important technical features of the I^{2}C bus and describe the special I^{2}C hardware interface of the 8XC751/752. We shall demonstrate with an example how the microcontroller can be programmed for a multimaster environment. The communications routines of the example are quite general, and can be ported to many applications—so we shall discuss in detail the software interface to these routines.

The description of the 8XC751 I^{2}C interface hardware and part of the general discussion of the I^{2}C bus is similar to Application Note AN422 which dealt with the microcontroller in a single-master environment. Most of the added discussions relate to the multimaster aspects of the bus. Additional information for the I^{2}C bus and the 83C751/752 Microcontroller can be found in the Philips Semiconductors Microcontroller Data Handbook (IC20).

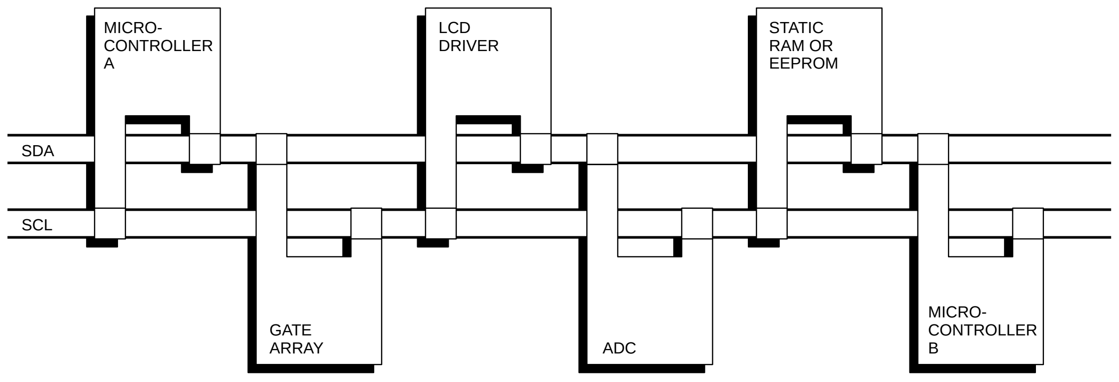

Figure 1. Example of an I^{2}C-bus Configuration

Using the 8XC751/752 in multimaster I^{2}C applications AN430

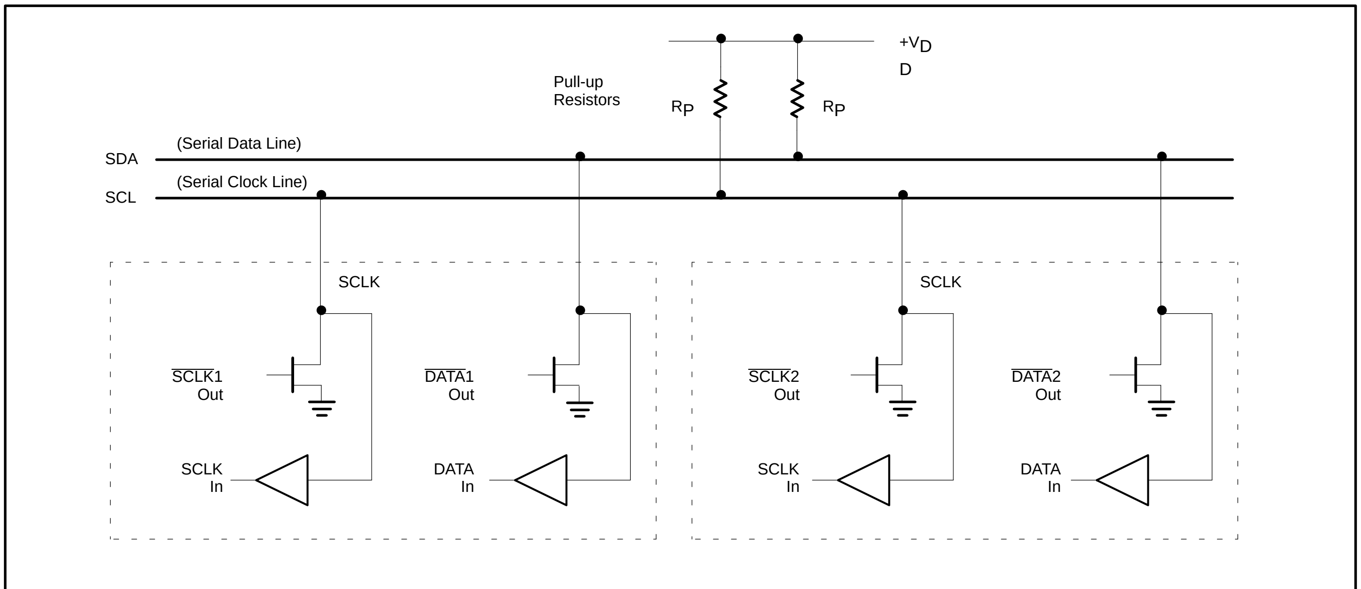

Figure 2. Connection of I^{2}C-bus Devices to the I^{2}C-bus

THE I^{2}C BUS

The two lines of the I^{2}C bus are a serial data line (SDA) and a serial clock line (SCL). A typical system configuration is shown in Figure 2. Each device is recognized by a unique address—whether it is a microcomputer, LCD driver, memory or keyboard interface—and can operate as either a transmitter or a receiver, depending on the function of the device. A device generating a message or data is a transmitter, and a device receiving the message or data is a receiver. Obviously, a passive function like an LCD driver could only be a receiver, while a microcontroller or a memory can both transmit and receive data.

Every device connected to the bus must have an open-drain or an open-collector output for both the data (SDA) and the clock (SCL) lines. Each one of the lines is connected to the positive supply via a common pull-up resistor (see Figure 2). This implements a wired-AND function, and each of the bus lines which will have the HIGH level only if all the output transistors tied to it are switched off.

Data on the I^{2}C bus can be transferred at a rate up to 100kbit/s. The number of devices connected to the bus is limited only by the maximum bus capacitance of 400pF. As different technology devices can be connected to the I^{2}C bus, the levels of the logical 0 (Low) and logical 1 (High) are not fixed and depend on the appropriate level of V_{DD}.

MASTERS AND SLAVES

When a data transfer takes place on the bus, a device can be either a master or a slave. The device which initiates the transfer, and generates the clock signals for this transfer is the master. At that time any device addressed is considered a slave. It is important to note that a master could be either a transmitter or a receiver: a master microcontroller may send data to a RAM acting as a transmitter, and then interrogate the RAM for its contents acting as a receiver—in both cases being the master initiating the transfer. In the same manner, a slave could be both a receiver and a transmitter.

The I^{2}C is a multimaster bus. It is possible to have in one system more than one device capable of initiating transfers and controlling the bus. A microcontroller may act as a master for one transfer, and then be the slave for another transfer, initiated by another processor on the network. The master/slave relationships on the bus are not permanent, and exist per transfer.

As more than one master may be connected to the bus it is possible that two devices will try to initiate transfer at the same time. Obviously, in order to eliminate bus collisions and communications chaos, an arbitration procedure is necessary. The I^{2}C design has an inherent arbitration and clock synchronization procedure relying on the wired-AND connection of the devices on the bus. In a typical multimaster system, a microcontroller program should allow it to gracefully switch between master and slave modes and preserve data integrity upon loss of arbitration.

DATA TRANSFERS

One data bit is transferred during each clock pulse (Figure 3). The data on the SDA line must remain stable during the HIGH period of the clock pulse in order to be valid. Changes in the data line at this time will be interpreted as control signals. A HIGH-to-LOW transition of the data line (SDA) while the clock signal (SCL) is HIGH indicates a Start condition, and a LOW-to-HIGH transition of the SDA while SCL is HIGH defines a Stop condition (Figure 4). The bus is considered to be busy after the Start condition and free again a certain time after the Stop condition. The Start and Stop conditions are always generated by the master.

The number of data bytes transferred between the Start and Stop condition from transmitter to receiver is not limited. Each byte, which must be eight bits long, is transferred serially with the most significant bit first, and is followed by an acknowledge bit (Figure 5). The clock pulse related to the acknowledge bit is generated by the master. The device that acknowledges has to pull down the SDA line during the acknowledge clock pulse, while the transmitting device releases the SDA line (HIGH) during this pulse (Figure 6).

A slave receiver must generate an acknowledge after the reception of each byte, and a master must generate one after the reception of each byte clocked out of the slave transmitter. If a receiving device cannot receive the data byte immediately, it can force the transmitter into a wait state by holding the clock line (SCL) LOW. When designing a system it is necessary to take into account cases when acknowledge is not received. This happens, for example, when the addressed device is busy in a real time operation. In such a case the master, after an appropriate "time-out", should abort the transfer by generating a Stop condition, allowing other transfers to take place. These "other transfers" could be initiated by other masters in a multimaster system or by this same master.

An exception to the "acknowledge after every byte" rule occurs when a master is a receiver: it must signal an end of data to the transmitter by NOT signalling an acknowledge on the last byte that has been clocked out of the slave. The acknowledge related clock, generated by the master, should still take place but the SDA line will not be pulled down. In order to indicate that this is an active and intentional lack of acknowledgement, we shall term this special condition as a "Negative ACK".

The bus design includes special provisions for interfacing to microprocessors which implement all the I^{2}C communications in software only—it is called "Slow Mode". When all the devices on the network have built-in I^{2}C hardware support the Slow Mode is irrelevant.

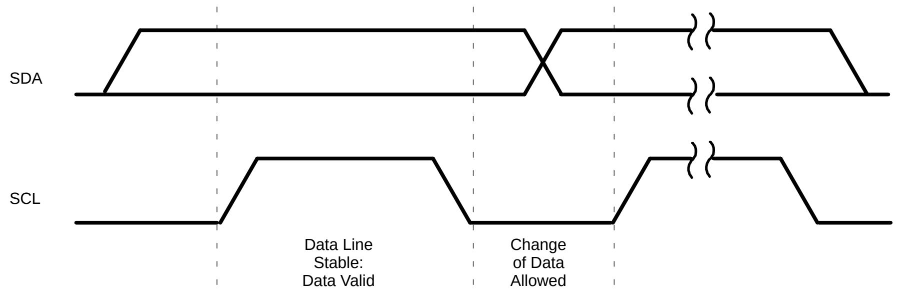

Figure 3. Bit Transfer on the I^{2}C Bus

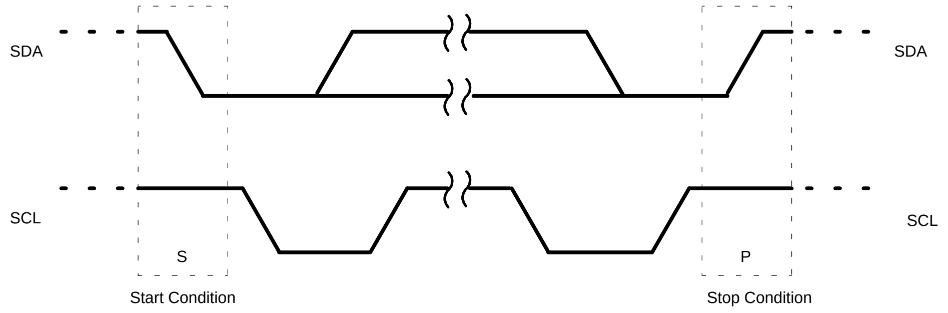

Figure 4. Start and Stop Conditions

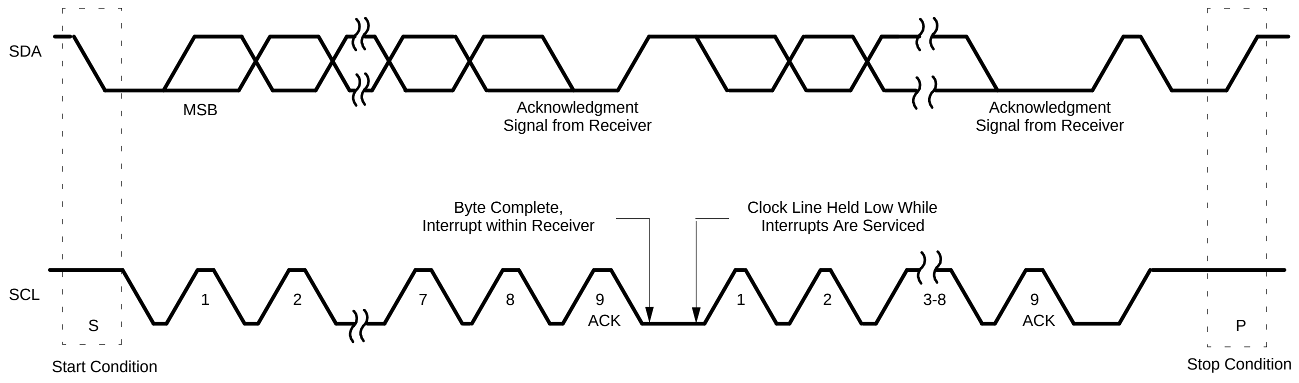
SDA
MSB
Acknowledgment Signal from Receiver
Acknowledgment Signal from Receiver
Byte Complete, Interrupt within Receiver
Clock Line Held Low While Interrupts Are Serviced
SCL
S
1
2
7
8
9 ACK
1
2
3-8
9 ACK
P
Start Condition
Stop Condition

Figure 5. Data Transfer on the I^{2}C Bus

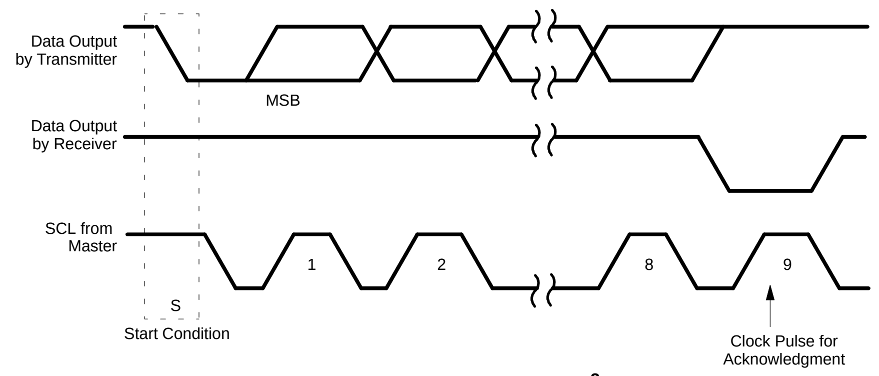

Figure 6. Acknowledge on the I^{2}C Bus

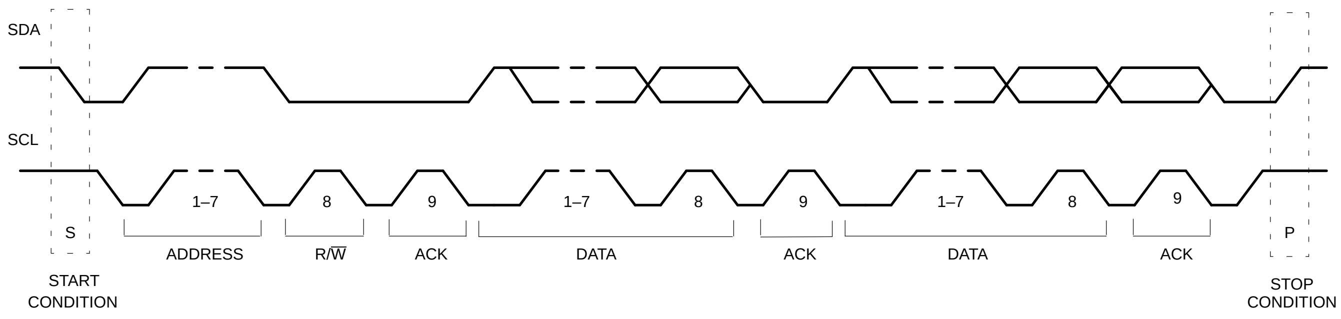

Figure 7. A Complete Data Transfer on the I^{2}C-Bus

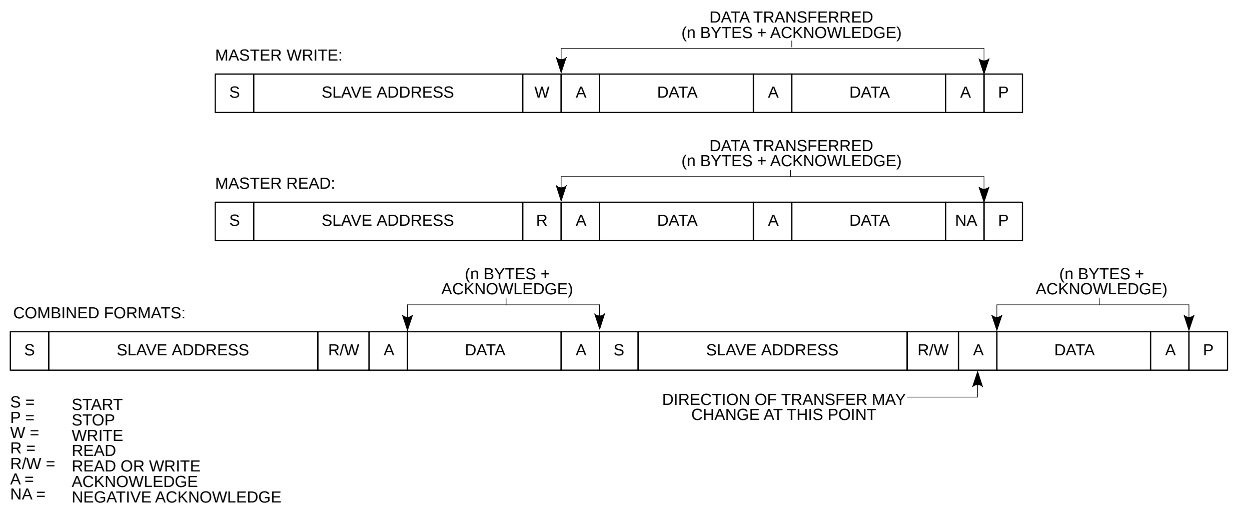
MASTER WRITE:
DATA TRANSFERRED (n BYTES + ACKNOWLEDGE)
S | SLAVE ADDRESS | W | A | DATA | A | DATA | A | P
MASTER READ:
DATA TRANSFERRED (n BYTES + ACKNOWLEDGE)
S | SLAVE ADDRESS | R | A | DATA | A | DATA | NA | P
COMBINED FORMATS:
(n BYTES + ACKNOWLEDGE)
(n BYTES + ACKNOWLEDGE)
S | SLAVE ADDRESS | R/W | A | DATA | A | S | SLAVE ADDRESS | R/W | A | DATA | A | P
DIRECTION OF TRANSFER MAY CHANGE AT THIS POINT
S = START
P = STOP
W = WRITE
R = READ
R/W = READ OR WRITE
A = ACKNOWLEDGE
NA = NEGATIVE ACKNOWLEDGE

Figure 8. I^{2}C Data Formats

ADDRESSING AND TRANSFER FORMATS

Each device on the bus has its own unique address. Before any data is transmitted on the bus, the master transmits on the bus the address of the slave of this transaction. A well-behaved slave, if it exists on the network, should of course acknowledge the master's addressing. The addressing is done with the first byte transmitted by the master after the Start condition.

An address on the network is seven bits long, appearing as the most significant bits of the address byte. The last bit is a direction (R/W) bit. A zero indicates that the master is transmitting (WRITE) and a one indicates that the master requests data (READ). A complete data transfer, comprised of an address byte indicating a WRITE and two data bytes is shown in Figure 7.

When an address is sent, each device in the system compares the first seven bits after the Start with its own address. If there is a match, the device will consider itself addressed by the master and will send an acknowledge. The device could also determine if in this transaction it is assigned the role of a slave receiver or slave transmitter, depending on the R/W bit.

Each node of the I^{2}C network has a unique seven bit address. The address of a microcontroller is, of course, fully programmable, while peripheral devices usually have fixed and programmable address portions. In addition to the "standard" addressing discussed here, the I^{2}C bus protocol allows for "general call" addressing and interfacing to CBUS devices.

When the master is communicating with one device only, data transfers follow the format of Figure 8 where the R/W bit could indicate either direction. After completing the transfer and issuing a Stop condition, if a master would like to address some other device on the network, it could start another transaction by issuing a new Start.

Another way for a master to communicate with several different devices would be by using a "repeated start". After the last byte of the transaction was transferred, including its acknowledge (or Negative ACK), the master issues again a Start, followed by address byte and data, without effecting a Stop. The master may communicate with a number of different devices, combining READS and WRITES. Only after the transfer with the last slave took place, the master issues a Stop and releases the bus. Possible data formats are demonstrated in Figure 8. Note that the repeated start allows for both change of a slave and a change of direction, without releasing the bus. We shall see later on that the change of direction feature can come in handy even when dealing with a single device.

In a single master system the repeated start mechanism is more efficient than terminating each transfer with a Stop and starting again. In a multimaster environment the determination of which format is more efficient could be more complicated, as when a master is using repeated starts it occupies the bus for a long time and prevents other devices from initiating transfers.

USE OF SUB-ADDRESSES

For some ICs on the I^{2}C bus the device address alone is not sufficient for effective communications and a mechanism for addressing the internals of the device is necessary. A typical example is addressing memories, when we want to access a specific word inside the device or a sequence of memory locations starting at a specific internal address.

A typical I^{2}C memory device like the PCF8570 RAM contains a built-in word address register that is incremented automatically after each read or written data byte. When a master communicates with the PCF8570 it must send a sub-address in the byte following the slave address byte. This sub-address is the internal address of the word the master wants to access for a single byte transfer or the beginning of a sequence of locations for a multi-byte transfer. A sub-address is an eight bit byte, unlike the device address it does not contain a direction (R/W) bit, and like any byte transferred on the bus it must be followed by an acknowledge.

A memory write cycle is shown in Figure 9(a). The Start is followed by a slave byte with the direction bit set to WRITE, a sub-address byte, a number of data bytes and a Stop signal. The sub-address is loaded into the word address memory. The data bytes which follow will be written one after the other starting with the sub-address location and the register is incremented automatically.

The memory read cycle (Figure 9(b)) commences in a similar manner with the master sending a slave address with the direction bit set to WRITE with a following sub-address. Then, in order to reverse the direction of the transfer, the master issues a repeated Start followed again by the memory device address, but this time with the direction bit set to READ. The data bytes starting at the internal sub-address will be clocked out of the device with each followed by a master-generated acknowledge. The last byte of the read cycle will be followed by a Negative ACK, signalling the end of transfer. The cycle is terminated by a Stop signal.

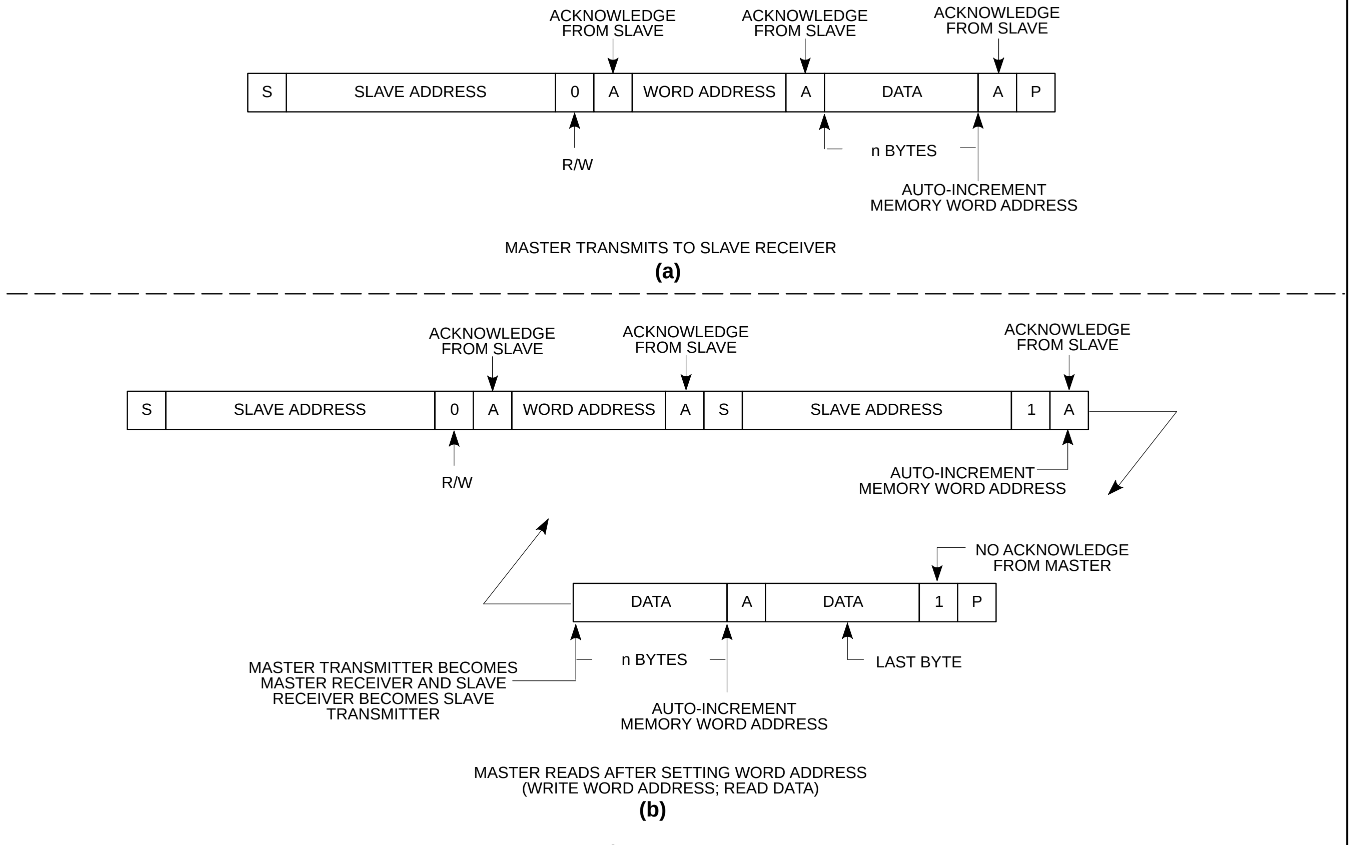
ACKNOWLEDGE FROM SLAVE
ACKNOWLEDGE FROM SLAVE
ACKNOWLEDGE FROM SLAVE
S
SLAVE ADDRESS
0
A
WORD ADDRESS
A
DATA
A
P
R/W
n BYTES
AUTO-INCREMENT MEMORY WORD ADDRESS
MASTER TRANSMITS TO SLAVE RECEIVER
(a)
ACKNOWLEDGE FROM SLAVE
ACKNOWLEDGE FROM SLAVE
ACKNOWLEDGE FROM SLAVE
S
SLAVE ADDRESS
0
A
WORD ADDRESS
A
S
SLAVE ADDRESS
1
A
R/W
AUTO-INCREMENT MEMORY WORD ADDRESS
NO ACKNOWLEDGE FROM MASTER
DATA
A
DATA
1
P
MASTER TRANSMITTER BECOMES MASTER RECEIVER AND SLAVE RECEIVER BECOMES SLAVE TRANSMITTER
n BYTES
LAST BYTE
AUTO-INCREMENT MEMORY WORD ADDRESS
MASTER READS AFTER SETTING WORD ADDRESS (WRITE WORD ADDRESS; READ DATA)
(b)

Figure 9. I^{2}C Sub-Address Usage

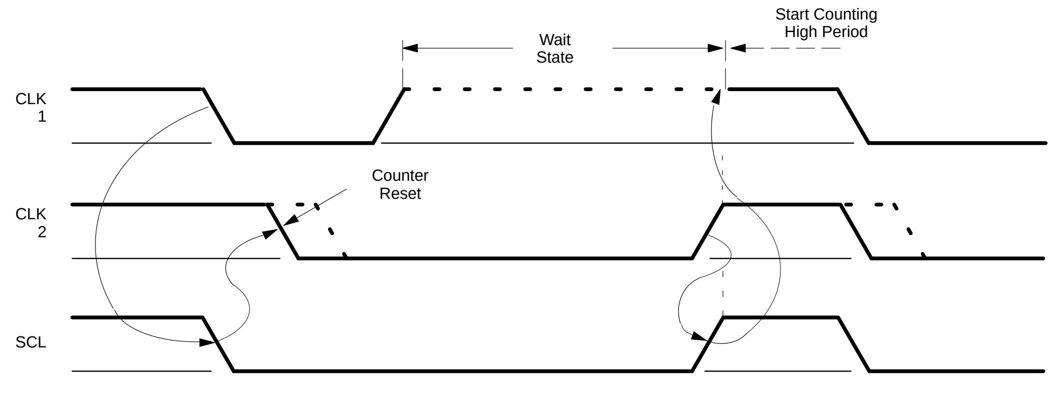

Figure 10. Clock Synchronization During the Arbitration Procedure

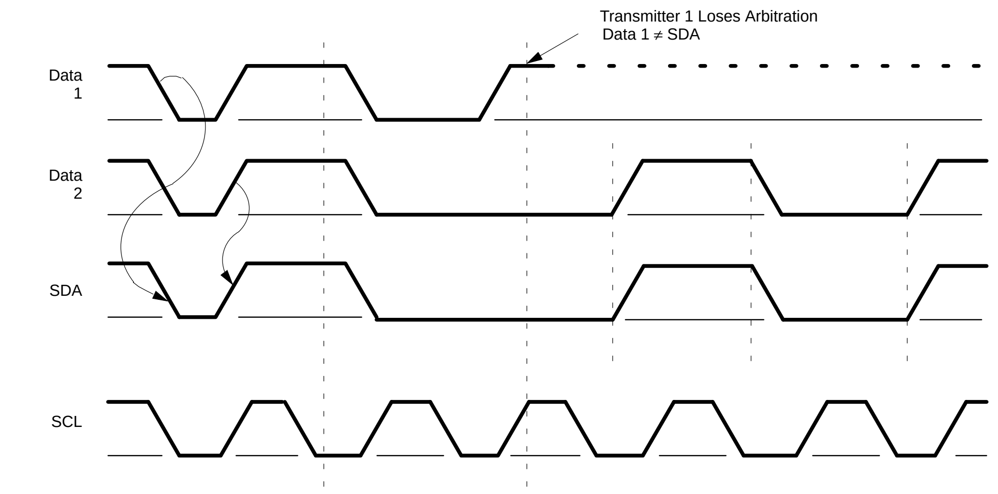

Figure 11. Arbitration Procedure of Two Masters

ARBITRATION IN A MULTIMASTER SYSTEM

The decision about which master has control over the I^{2}C bus is based solely on the address and data sent by competing masters, and there is no central master or any order of device priority on the bus. Any device connected to the I^{2}C bus is allowed to become a master, but devices are not supposed to “steal” the bus from other devices when a transfer is in process. If a device wishing to be a Master is aware that a transaction (initiated by another master) is taking place, it will wait until the transfer is concluded with a Stop condition on the bus—and only then try to seize it by sending its own Start. It is possible, however, that two or more masters may want to start a transfer at exactly the same moment. A scenario that may happen quite frequently in a loaded system: two devices are waiting for a long transaction to be completed, and simultaneously try to get the bus when detecting the Stop condition. An arbitration procedure synchronizes the different clocks, ensuring that the data is not corrupted, and causes all masters except one to withdraw from the bus, so only one master will control the transfer. This procedure applies only when masters initiate transfers simultaneously.

The clock synchronization, illustrated in Figure 10, ensures that only one defined clock is generated on the bus. It occurs naturally, as a result of the wired-AND property of the SCL line. Suppose two masters want to initiate a transfer on the bus. Clk1 and Clk2 in Figure 10 illustrate the desired clock outputs of each device, which would actually occur on the bus if each were the only master. The SCL waveform is the resulting wired-AND of the two clocks. The device that pulls the SCL down first will succeed. The other masters continuously monitor the clock line, and reset their internal clock counter to start counting their own Low clock period. This way, the first falling edge will synchronize all clock generators to the beginning of the Low time.

Once a device clock has gone Low it will hold the SCL line in this state until its internal clock High state is reached, and then will release the line. The Low to High change in this device will not change the state of the SCL line if another device, which is still within its Low period, is pulling down the line. This way, SCL will be held Low by the device with the longest Low period. A master that has finished its Low time earlier will enter a wait state until SCL is released by the slowest master and goes high. Upon the rising edge of SCL all masters start counting their High period, the first device to complete its High period will pull the SCL Low. In this way a single, synchronized clock is generated on the bus where the rising edge is being defined by the slowest master and the falling edge by the fastest master.

Arbitration between masters takes place on the SDA line. A master which tries to transmit a High while another device transmits a Low will withdraw, shutting off its data output stage and not interfering with the transfer until a Stop condition is detected. Due to the wired-AND property of the SDA line, a device “knows” that it lost arbitration by the fact that the Low SDA is different than its desired High output. Arbitration starts by comparing the address bits. When masters transmit different addresses the one transmitting the address with the lowest binary value wins. If all masters in arbitration transmit to the same address, arbitration continues into the comparison of data. Figure 11 illustrates the arbitration process between two masters.

By definition, the transfer that forces the wired-AND result is the one that wins the arbitration, so the address and data of a winning device are not corrupted and no information is lost in the arbitration process. A master losing arbitration may generate clock pulses until the end of the byte. Thus it may affect the clock speed, but not the data on the bus.

If a master loses arbitration during the addressing stage it is possible that the winning master is trying to address it. In an efficient design, the losing master should switch immediately to its slave receiver mode, receive the data transmitted and acknowledge it—otherwise the message will have to be re-transmitted or is lost. A well designed master will take into account “illegal” protocol situations and will determine that it lost arbitration when it detects a Stop or a Start which are not synchronized with its own transmission. Electrical interference or a malfunctioning device may cause such a situation which actually corrupts the message transfer.

HANDSHAKE BY CLOCK SYNCHRONIZATION

The clock synchronization mechanism as described above actually implements a handshake mechanism, enabling receiving devices to “slow down” fast transfers when necessary.

On the bit level, a slow slave device like a microcontroller that does not have hardware I^{2}C interface port, can extend each clock period and slow down the bus clock. The speed of any master is adapted to the operating rate of this device as long as it is active on the bus.

On the byte level the synchronization mechanism takes effect as a “handshake” mechanism when a slave device that was fast enough to receive or transmit a byte still needs extra time to store the received byte or prepare the next byte for transmission. The slave can hold the SCL line low after the reception and acknowledge of a byte, thus forcing the Master into a wait state—until the slave is ready for the next transfer.

8XC751 I^{2}C HARDWARE

The on-chip I^{2}C bus hardware support of the 8XC751 allows operation on the bus at full speed and simplifies the software needed for effective communications on the network. The hardware activates and monitors the SDA and SCL lines, performs the necessary arbitration and framing error checks, and takes care of clock stretching and synchronization. The hardware support includes a bus timeout timer, called Timer I. The hardware is synchronized to the software either through polled loops or interrupts.

Two of the port 0 pins are multi-functional. When the I^{2}C is active, the pin associated with P0.0 functions as SCL, and the pin associated with P0.1 functions as SDA. These pins have an open drain output.

Two of the five interrupt sources may be used for I^{2}C support. The I^{2}C interrupt is enabled by the EI2 flag of the interrupt enable register, and its service routine should start at address 023h. An I^{2}C interrupt is usually requested (if enabled) when a rising edge of SCL indicates new data on the bus or a special condition occurs: Start, Stop or arbitration loss. The interrupt is induced by the ATN flag, (see below for the conditions for setting this flag). The Timer I overflow interrupt is enabled by the ETI flag, and the service routine starts at 01Bh.

The I^{2}C port is controlled through four special function registers: I^{2}C Control (I2CON), I^{2}C Configuration (I2CFG), I^{2}C Data (I2DAT) and I^{2}C Status (I2STA). The register addresses are shown in the 8XC751 section of the Philips Semiconductors Microcontroller Data Handbook (IC20). Although the following discussion of the hardware and register details is not complete, it should give a better understanding of the programming examples.

Timer I

In I^{2}C applications, Timer I is dedicated to the port timing generation and bus monitoring. In non-I^{2}C applications, it is available for use as a fixed rate timer.

For the bus monitoring function, Timer I is being used as a “watchdog timer” for bus hang-ups. It creates an interrupt when the SCL line stays in one state for an extended period of time between a Start condition and a following Stop condition. SCL “stuck low” indicates a faulty master or slave. SCL “stuck high” may mean a faulty device or that noise induced into the I^{2}C caused all masters to withdraw from the I^{2}C arbitration.

The time-out interval of Timer I is fixed: it carries out and interrupts (if enabled) when about 1024 machine cycles have elapsed since a change on SCL within a frame. In other words, whenever I^{2}C is active we let Timer I run, but clear it whenever a frame is not in progress (reset or Stop occurred more recently than the last Start condition) or SCL changes within a frame. (Note: we wrote “about 1024 machine cycles” for the sake of accuracy—this number may slightly change according to the setting of the CT0 and CT1 bits mentioned below. In any case, the exact number of cycles for a time out does not have any practical significance).

In addition to the interrupt upon Timer I overflow, the I^{2}C port hardware is reset. This is useful for multiple master systems in situations where this same 8XC751 caused the bus hang-up due to a lack of software response. SCL will be released and I^{2}C operation between other devices could continue.

I2CON Register

The I^{2}C Control register can be read or written to (see Figure 12).

When writing to the I2CON register one should use bit masks as demonstrated in the examples. Trying to clear or set the bits in the register using the bit addressing capabilities of the 8XC751 may lead to undesirable results. The reason is that a command like CLRB reads the register, sets the bit and writes it back—and the write-back may affect other bits.

I2CFG Register

The configuration register is a read/write register (see Figure 13).

I2DAT Register

The I^{2}C data register is a read/write register, where the msb represents the data received or data to be sent. The other seven bits are read as 0 (see Figure 14).

I2CSTA Register

The I^{2}C STAtus Register is a read-only register reflecting the internal status of the I^{2}C interface hardware (see Figure 15).

Transmit Active State

The transmit active state—Xmit Active—is an internal state in the I^{2}C interface that is affected by the I^{2}C registers as explained above. The I^{2}C interface will only drive the SDA line low when Xmit Active is set. Xmit Active is set by writing the I2DAT register or by writing I2CON with XSTR = 1 or XSTP = 1. The ARL bit will be set to 1 only when Xmit Active is set—in such a case Xmit Active will be automatically reset upon ARL. Xmit Active is cleared by writing 1 to CXA at I2CON register or by reading the I2DAT register.

I2CON READ	RDAT	ATN	DRDY	ARL	STR	STP	MASTER	—

RDAT	Received DATa bit. The value of SDA latched by the rising edge of SCL. Its contents is identical to RDAT in I2DAT register. Reading the received data here allows doing so without clearing DRDY and releasing SCL.
ATN	An “ATteNsion” flag, set when any one of DRDY, ARL, STR or STP is set. This flag allows a single bit testing for terminating “wait loops”, indicating a meaningful event on the bus. This same flag actually activates the I^{2}C interrupt request.
DRDY	Data ReaDY flag, set by a rising edge of SCL when I^{2}C is active, except at an idle slave. This flag is cleared by reading or writing the I2DAT register, or by writing a 1 to CDR (same address, I2CON write).
ARL	ARbitration Loss flag. Indicates that this device lost the arbitraion while trying to take control of the bus.
STR	STaRt flag is set when a Start condition is detected, except at an idle slave.
STP	SToP flag is set when a Stop condition is detected, except at an idle slave.
MASTER	This flag is set when the controller is a bus master (or a potential master, prior to arbitration).

I2CON WRITE	CXA	IDLE	CDR	CARL	CSTR	CSTP	XSTR	XSTP

CXA	“Clear Xmit Active”. Writing a 1 to CXA clears the internal transmit-active state.
IDLE	Setting this bit will cause a slave to idle—ignore the I^{2}C until the next Start is detected. If the software sets the MASTRQ flag the device may stop idling by turning into a master.
CDR	Clear Data Ready—clears the DRDY flag.
CARL	Clear Arbitration Loss—clears ARL flag.
CSTR	Clear STaRt—clear STR flag
CSTP	Clear STop—clear STP flag.
XSTR	“Xmit repeated STaRt”. writing a 1 to this bit causes the hardware to issue a Repeated Start signal. A side effect will be setting the internal Xmit Active state. This should be used only when the device is a master.
XSTP	“Xmit SToP”. Issue a Stop condition. The Xmit Active state is being set.

Figure 12. I2CON Register

SLAVEN	MASTREQ	CLRTI	TIRUN	—	—	CT1	CT0

SLAVEN	Writing a 1 to this flag enables the slave functions of the I^{2}C interface.
MASTREQ	Request control of the bus as a master.
CLRTI	Clear the Timer I interrupt flag. This bit is always read as 0.
TIRUN	Writing a 1 will let Timer I run. When I^{2}C is active, it will run only inside frames, and will be cleared by SCL transitions, Start and Stop. Writing a 0 will stop and clear the timer.
CT1, CT0	These bits should be programmed according to the frequency of the crystal oscillator used in the hardware. They control a frequency devider which determines the timing on the bus, and are used to optimize performance at different oscillator speeds.

Figure 13. I2CFG Register

I2DAT READ	RDAT	—	—	—	—	—	—	—

RDAT	Received DATa bit, captured from SDA every rising edge of SCL. Reading I2DAT clears DRDY and Xmit Active state. If it is necesary to read the data without affecting the flags, it can be read out of RDAT in I2CON register.

I2DAT WRITE	XDAT	—	—	—	—	—	—	—

XDAT	Xmit DATa bit. Writing XDAT determines the data for the next bit to be transmitted on the I^{2}C bus. Writing I2DAT also clears DRDY and sets the Xmit Active state.

Figure 14. I2DAT Register

I2CSTA READ	IDLE	XDATA	XACTV	MAKSTR	MAKSTP	XSTR	XSTP	—

IDLE	Indicates when the I^{2}C hardware is in the Idle mode.
XDATA	Reflects the contents of the I^{2}C transmitter buffer.
XACTV	Indicates that the I^{2}C transmitter is active.
MAKSTR	Indicates that the hardware is effecting a Start.
MAKSTP	Indicates that the hardware is effecting a Stop.
XSTR	Hardware effecting a Repeated Start.
XSTP	Hardware effecting a Stop.

Figure 15. I2CSTA Register

I^{2}C COMMUNICATIONS SOFTWARE

The software listing demonstrates programming the 8XC751/752 for a multimaster I^{2}C environment where the device can be both a Master or a Slave responding to other Masters on the I^{2}C network. The bulk of the software is communications routines which are not only for demonstration but could be ported to other user programs with minimal or no modifications. The routines are quite general and could be useful in most applications. We have tried to design a well-defined software interface, enabling most users to copy the routines as they are, modifying only the pre-defined interface elements to fit the specific applications. We encourage users to use the routines without modifications whenever possible, as the lower levels of the hardware-software integration could be quite involved.

The rest of this application note will relate to the programming example. We shall discuss the general operation of the routines and how they are integrated into an application. Then we shall describe in detail all the software interface elements and how to use them.

I^{2}C COMMUNICATIONS ROUTINES—OVERVIEW

In order to function well in a multimaster environment the microcontroller must be able to take control of the I^{2}C bus as a Master, “tolerate” message transactions between other Masters and other devices, and respond efficiently as a Slave to other bus Masters. The communications routines should allow a Master “graceful” recovery from an arbitration loss and other situations when a message transaction is not completed, allowing for communication re-tries.

For Slave operation the microcontroller must be interrupt driven relative to an I^{2}C frame Start, as any Master on the bus could request a transaction at any moment, not synchronized to the application program executing on the controller. An interrupt service routine monitors the address transmitted on the bus. When the microcontroller is addressed it takes care to either read the data from the bus into a buffer or write buffer data onto the bus. When such a transaction is successfully completed, one of several “Slave Event Routines” is called prior to returning to the main application program. Such an “Event Routine” is a part of the application, allowing an immediate response to the data received, or the fact that data was transmitted to a requesting Master. This allows “synchronization” of the application to a “slave” bus transaction. Typical uses of the Event Routine mechanism will be a computation based on new data, or re-loading the transmit buffer with new data getting ready for the next random request. The actual Event Routines will be programmed differently for different applications, but the names and the calls will remain the same as long as the communications routines are left unmodified.

A transaction as a Master is initiated by the application program. Our implementation uses the interrupt mechanism for the Master communications as well. The application issues a request for the bus by setting the MASTRQ bit of the I^{2}C port control, and when the bus is available an interrupt occurs. This way, if the bus is free there will be an immediate response. If the bus is busy, the application may go on executing (if so programmed) until this controller can get control of the bus. When the microcontroller gets mastership of the bus it initiates a bus transaction according to “directives” set by the application program. The most important directives are the address (and subaddress if relevant) of the slave device addressed, and the length of the message to be transmitted or received.

When a Master transaction is concluded, a Master Event Routine (called MastNext) is called to perform whatever task the application demands. As with the Slave Event Routines it will typically respond to a successful transmission or reception of data. In addition, it could handle situations where a slave does not respond at all, or does not acknowledge a data byte (thus causing data transfer to terminate). A program might react to the fact that a slave does not respond by re-trying to communicate at a later time, by issuing a message to another peripheral device or just ignoring it. The handling of such cases is application dependent, and should be programmed into the routine called “MastNext”. The MastNext routine is invoked when the Master terminates the transaction “willingly”, but not upon arbitration loss.

The microcontroller operating as a bus Master may lose arbitration to another Master which happens when two Masters transmit in synchronization, commencing with the same Start signal. If arbitration is lost while transmitting or receiving data, our processor withdraws from the bus and turns itself into a slave—an active Slave upon a Start, or returning to the calling program as an idle slave. When the arbitration loss occurs while transmitting an address, our processor turns itself immediately into an active slave, “listening” to the rest of the address transmitted by the new Master. If our processor reads its own address from the bus (as transmitted by the new Master) our processor responds as a willful slave. If this mechanism would not have been implemented, there could be potential inefficiency when a device that happened to be synchronized to another Master loses arbitration, but is not able to respond to the winning device.

Another situation for arbitration loss could be a bus exception resulting from a device operating not according to the bus protocol or interference on the bus lines. In addition to “regular” arbitration loss detected with the ARL hardware flag, such a situation may occur with detecting a Start or a Stop in the middle of transmitting an address or data byte. In such a situation the microcontroller withdraws from the bus as well—active Slave upon a Start detection, or returning as an idle slave in other cases.

When a Master transaction is terminated by an arbitration loss, the Master Request flag (MASTRQ) of the hardware I^2C port remains in effect. As a result when the bus gets free, our device will take control, issue a Start, and the transaction that was cut will start again. This restart will happen automatically, without any application involvement (unlike non-acknowledgement, where the MastNext routine determines what shall be done).

The I^2C communications routines are structured as an interrupt service routine responding to an I^2C port interrupt upon a frame Start. Within a frame the I^2C processing is continuous, where the I^2C port is polled for hardware response, and the I2C interrupts are disabled. Other interrupts are enabled during the service routine. The set-up requirements from the mainline program are minimal, and interfacing is done via RAM buffers and some pre defined RAM locations. The lower level interface with the hardware is done inside the service routine, and can typically be ignored by the application programmer.

BUS WATCHDOG AND ERROR RECOVERY

A malfunctioning device (in hardware or software) may hold the SCL line low, thus causing the bus to be “stuck”. It might even be possible that a transient protocol violation (due to hardware interference, such as a device turning on) may cause some devices (non programmable, or even microcontrollers which were not carefully programmed) to hold the bus. Since within a frame the bus is software-polled, a “stuck” bus might cause the application software to “hang forever”. Here the TIMERI watchdog comes to the rescue, interrupting when there is no bus activity for a long period of time.

When the I^2C service routine is interrupted by the watchdog timer, the processing of the current frame is not completed and the event routines are not called. The software returns to execute the mainline application, and will be interrupted again for the next frame (next Start, received as a slave or induced as a Master). A status flag and a counter report on the watchdog interrupt, so the application program can be made to inhibit the I^2C port if there are too many occurrences of a “hanging” bus.

Bus protocol errors and “hangups” might be an issue in systems which are susceptible to noise, temporary bus line shorts, “hot plug in” of devices or even erroneously programmed devices—and a “fail safe” controller program should be able to detect bus problems and possibly assist in resolving them. The RECOVER routine resets the I^2C interface of the microcontroller, and attempts to release some other devices on the bus by toggling the clock line. The I^2C interface of the 8XC751 is reset by letting TimerI run and expire, since this circuitry does not feature a software controlled reset. This “extreme” measure is needed in some cases of bus protocol violation.

The bus and interface circuit recovery routine can be automatically invoked whenever TimerI detects a timeout. In addition, for systems where potential bus failures are a concern and reliability is an issue, one may implement mechanisms to invoke bus and interface recovery from the application code. This may help in cases where the bus gets “stuck” when there is no I^2C frame in progress. In such an instance the watchdog timer will not give any timeout indications, as it has not been activated. Another case emanates from a design peculiarity of the interface circuitry on the 8XC751: if the SCL line is externally grounded when there is a Start condition, this Start might be ignored, and the watchdog may not be activated. Our programming example deals with potential failures by testing for transaction completion and retrying transmissions when necessary (these are explicit retries, in addition to an “automatic” retry after a Master's arbitration loss, invoked by the MASTRQ bit). Too many transmission failures activate the RECOVER routine.

I^2C COMMUNICATIONS ROUTINES—INTERFACE

The I^2C service routine deals with the transmission and reception of messages, without any concern for the contents of the message. In order to provide a general interface for different applications the data is transferred via buffers. The service routine does not have to “know” where the data goes to or comes from—as long as the application program specifies the required pointers for these buffers. The interface to the actual application (which “cares” about message contents, timing, addressing and so forth) is done in a well defined manner, allowing usage of the same service routine with different application programs.

The interface is carried out with the use of buffers, pre-defined names for Application Event Routines, interface RAM locations for transferring parameters, pointers and flags, and constants. A more detailed discussion of the interface follows.

Buffers

There are three buffers for data transfers between the I^2C bus and the application program.

MasBuf is used for Master transmission and reception. The number of data bytes for each Master message—reception or transmission, is specified by the memory location MASTCNT. The value in MASTCNT should be less than the length of MasBuf. For Master transmission the message is placed in MasBuf before the transmission is initiated. In Master reception, the received message will be contained in the same buffer. There is only one Master message transaction occurring at the same time, so we may use the same buffer both for transmission and reception.

For Slave operation we must accommodate data transfers which may come randomly, asynchronous to each other or to possible operation of the same device as a Master. Therefore it is necessary to allocate additional RAM area as buffers dedicated to Slave operation: SRcvBuf for receiving data, STxBuf for transmission.

The length of the Slave receive buffer is defined by the symbol RBufLen. It is used by the code for protection, avoiding overwriting RAM beyond the allocated buffer size in case a Master sends a message which is too long. There is no need for RAM protection for transmission, but the Master should not request more data than STxBuf can supply.

Interface RAM Locations

RAM location MyAddr contains the address of this processor.

Status flag MSGSTAT is used for reporting to the application on I^2C communications status—mainly on the successful, or unsuccessful, completion of a message transaction. The contents of MSGSTAT may be used by the mainline application code or by the Event Routines. The different codes that could be placed by the I^2C service routine are described later in the text. When the message processing commences, a code indicating Slave or Master processing is inserted to MSGSTAT, and is updated as we go along. There could be many applications that will not need to use MSGSTAT contents, as the very fact of calling a certain event routine implies completion of a processing stage.

For Master transactions, in addition to the data buffer MasBuf, there are several RAM locations into which the application inserts Master message “directives”. These directives provide the service routine with the information necessary to carry out the next Master transaction. The one byte RAM locations used for directives are DESTADRW, DESSUBAD, MASTCNT and MASCMD.

DESTADRW contains the destination slave address in bits 7-1, while bit 0 is the R/W bit. Bit 0 contains 0 for a Write operation (the message is to be transmitted to the salve) and 1 for a Read operation (message is being read from the slave and received by this Master).

DESSUBAD contains the 8 bit sub-address of the slave, if necessary. For transactions without a sub-address, the contents of DESSUBAD is ignored.

MASTCNT contains the number of data bytes in the message to be sent from or received into MasBuf. This number should not be bigger than the length of MasBuf.

MASCMD byte contains the bit flags SUBADD, RPSTRT and SETMRQ. SUBADD is 0 (cleared) for a message with a regular address, and 1 (set) when a subaddress is required. When SUBADD is set, the service routine takes care of all the protocol required for sub-addressing, which includes a Repeated Start for Read operations. A message with a subaddress is considered to be a single message, even if it includes a Repeated Start.

The RPSTRT and SETMRQ are kept cleared in regular applications, and will be used only for “tailoring” the bus transfers in special cases. When RPSTRT is cleared the message will terminate, as usually required, with a Stop. When RPSTRT is set a Repeated Start will be sent on the bus, and Master operation will resume. The RPSTRT directive relates to terminating the message after all the data was transferred, and not to the mandatory Repeated Start in the middle of sub-addressed Read operation. A single message with a subaddress will typically have RPSTRT cleared. SETMRQ indicates what will be loaded into the MASTRQ flag of the hardware when Stop is transmitted. Typically it will be cleared. When SETMRQ is 1, MASTRQ will be set, thus trying to issue a new Start immediately following the Stop. In such a case the service routine will not return upon Stop, but will continue as a Master.

TITOCNT is used to count time-outs of the watchdog timer. Whenever such a timeout invokes the TIMER I interrupt service routine the contents of the location TITOCNT are incremented, and the timeout is reported in MSGSTAT. The count is saturated at 0FFh. This mechanism may be used in an application that is very much “concerned” with potential bus failures, allowing some type of “failure monitoring” by the application even for Slave transactions.

APPLICATION EVENT ROUTINES

The service routine calls Event Routines with pre-defined names (Figure 16), and these routines must be provided by the application program. The actual code of the routines will differ from application to application, but the routine names are being kept the same.

These routines are being called when successful processing of a message (send or receive) is completed. The routines may perform whatever action the application was designed for, which is not necessarily related to the I^2C communications mechanism. In addition, the routines may perform the data interface tasks for the I^2C port, like emptying buffers from received data or preparing the next message by setting up the buffers.

The mechanism of calling the event routines out of the service routine allows an immediate reaction to the event of message processing completion, before any new activity happens on the bus. In some simple applications this may not be necessary. For example, one may have a main program for a slave which is just a wait loop monitoring a flag set by the service routine when a message transfer, initiated by some master, is completed. In such a case the application could react to the message completion after the interrupt service routine returns. However, in the general case this will not be sufficient. An example could be a slave with an application which is constantly busy doing another task, in an environment where the communication requests on the I^2C bus are frequent. If there is a new message request shortly after the current message is completed, having to wait for the application until it “has time” may result in not reacting, or sending the same data again, or overwriting the received data in the buffer. Another obvious case demanding event routine calls is a Master sending different messages with a Repeated Start—the new data for the following message must be prepared in the interrupt service routine as the current message is completed (there is no return from interrupt prior to the new data transmission).

The programmer has the flexibility to decide where to prepare the next message according to the requirements of the application. This can be done after return from the event routine, in the application code after the return from interrupt, or a combination of both, where the time critical events are performed in the event routines. The application may monitor the MSGSTAT flag for message processing completion. If the event routines are not used, it is recommended to simply code them as a “RET” instruction, thus turning them into dummy routines (this an easier and better practice than changing the service routine itself, eliminating the calls).

Master Event Routine:

MastNext

This routine is called by the service routine when the processing of the current Master message is completed. For an indication on the type of message processing completion, MastNext may inspect the contents of MSGSTAT RAM location.

When MastNext is called, MSGSTAT will contain one of the following codes for message processing completion:

MRCVED (= 21h)—a complete message (with number of data bytes indicated by MASTCNT) was received from the slave.

MTXED (= 22h)—the number of data bytes indicated by MASTCNT were successfully sent and acknowledged by the slave.

MTXNAK (= 23h)—the slave did not acknowledge a data byte of the message, even though it had acknowledged its address. The message transmission was terminated upon the NAK.

MTXNOSLV (= 24h)—no slave acknowledged the address indicated by memory location DESTADDR.

The MastNext routine may perform any task(s) necessary for the application. Data handling tasks will typically be dependent on the MSGSTAT indication. One possible task could be setting the directives for the next message. The necessity for executing this task here (versus the main-line code initiating the transfer) is of course application dependent.

Slave Event Routines:

These routines are called when a message transaction as a slave has been completed. In many cases it could be important to utilize the calls to such routines as the requests for message transactions as a slave can come randomly, asynchronous to the application program. The application may demand that new data coming in should immediately

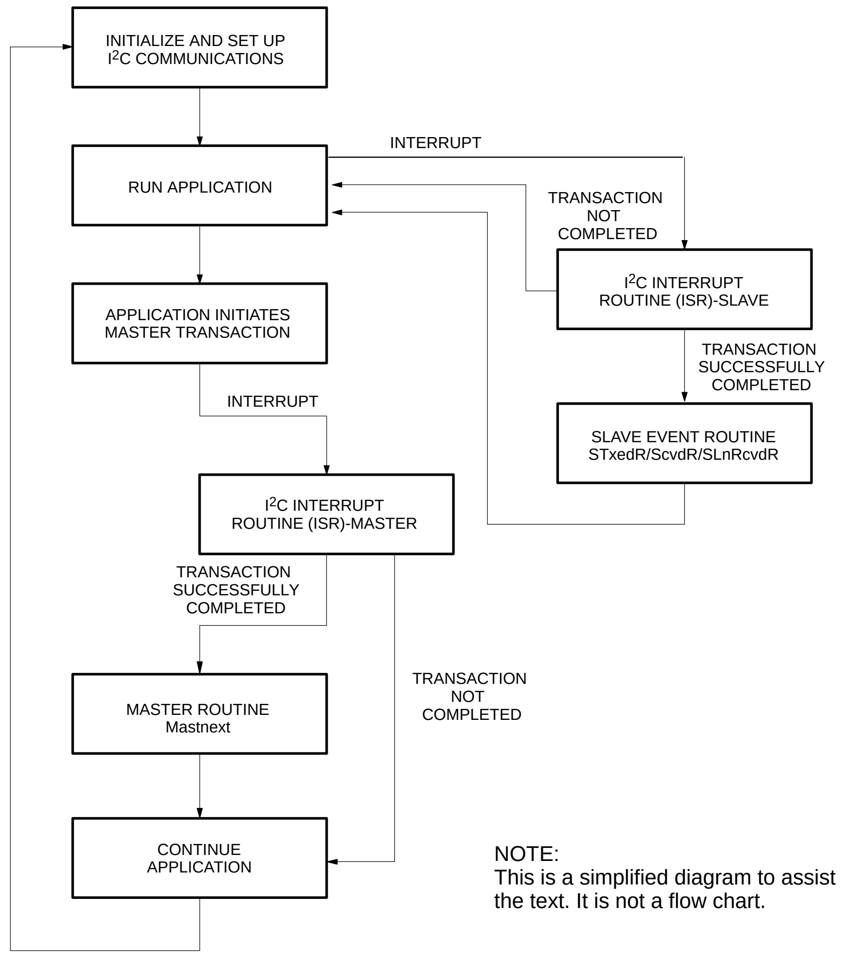
INITIALIZE AND SET UP I2C COMMUNICATIONS
RUN APPLICATION
INTERRUPT
TRANSACTION NOT COMPLETED
I2C INTERRUPT ROUTINE (ISR)-SLAVE
TRANSACTION SUCCESSFULLY COMPLETED
SLAVE EVENT ROUTINE STxedR/ScvdR/SLnRcvdR
APPLICATION INITIATES MASTER TRANSACTION
INTERRUPT
I2C INTERRUPT ROUTINE (ISR)-MASTER
TRANSACTION SUCCESSFULLY COMPLETED
MASTER ROUTINE Mastnext
TRANSACTION NOT COMPLETED
CONTINUE APPLICATION
NOTE:
This is a simplified diagram to assist the text. It is not a flow chart.

Figure 16. Typical Communications Scenario—A Simplified Diagram

initiate some tasks (e.g. control an output port)—and the event routine can be used to process the result of the slave interrupt.

In most cases it will be necessary for a slave to react immediately to a message received simply in order not to lose the data. As a new message may come randomly, it may overwrite the reception buffer before the data has been transferred out of it or acted upon.

For applications in which the reaction for slave events is performed after the return from the service routine, the event is reported by placing an appropriate code in the MSGSTAT flag. The programmer may use event routines, other mainline routines inspecting MSGSTAT, or both. If the event routines are not used, it is recommended to code them as a “RET” instruction.

SRcvdR:
Called by the service routine when a new, complete message has been received into SRcvBuf. When SRcvdR is called, R1 points to the address of the last byte received into the buffer. In a typical application SRcvdR will transfer the new data out of SRcvBuf, so it will not be written over by a subsequent slave reception.

The equivalent MSGSTAT indication for this event is SRCVD (= 11h).

SLnRcvdR:
Called when a slave message has been received into SRcvBuf, but the message was longer than the SRcvBuf buffer (as specified by RbufLen).

The equivalent MSGSTAT indication for this event is SRLNG (= 12h).

If the program is supposed to react to a too long a message the same way as to a message that can be contained in the buffer, one may code SLnRcvdR simply as a call to SRcvdR.

STXedR:
Called by the service routine when data has been transmitted out of the slave STxBuf buffer according to a master’s request. This routine may insert new data into the buffer, preparing it for the next slave transmission.

The equivalent MSGSTAT indication for this event is STXED (= 13h).

Note that we do not have a separate routine for the case that the master requested too many bytes—more than STxBuf length—and we sent out meaningless bytes. It is the master’s responsibility to specify the message length, and it should be able to request messages with the appropriate length from each slave on the bus.

SRErrR:
This routine relates more to bus communications than to the application itself.

It can be called when we positively detect a bus error upon reception as a slave, in case the application is supposed to know about it. In most cases this call will not be used, as dealing with bus communications difficulties is usually left to the Master.

Just prior to calling SRErrR, the code SRERR (= 14h) is placed in MSGSTAT.

0Completion Routine: I2CDONE

This routine is called every time, before returning from the I^{2}C interrupt service routine, whether the transaction was successful or not. It can be used to "safely" monitor MSGSTAT without any risk of a new interrupt modifying the current indication. Simple application programs will not make use of this routine. A more sophisticated application implementing a fail-safe communications protocol may use it to count errors of a certain type in order to determine a recovery scheme. In our programming example, I2CDONE inhibits I^{2}C interrupts when it is evident that as a result of protocol errors interrupts are not caused by legitimate Starts.

CONSTANTS

RBufLen—the length of SRcvBuf, the slave receive buffer. This constant may be used both by the I^{2}C routines and the application program, and it is the responsibility of the application programmer to define the correct buffer length.

MYNUM—This ROM constant is dependent on the application environment. It is a small integer defining a "serial number" of the node, out of all the processors running the same code. This constant is used only when recovering from a timeout, in order to "de-synchronize" masters from each other when trying to recover the bus.

CTVAL1 is a constant defined in ROM. It is used by the application code portion which initializes the I^{2}C, for loading CT0 and CT1 with a value appropriate for the crystal being used.

MYADDR1 is a ROM constant containing the address of the processor's I^{2}C node. This value is used by the application demo to load the RAM location MyAddr.

USING THE COMMUNICATIONS SUBROUTINES

In order to use the I^{2}C Communications Routines an application program should take care of the following:

– Upon initialization, load bits CT1, CT0 of I2CFG register according to the clock crystal used (refer to the table of CT1, CT0 values in the 8XC751 section of the Philips Semiconductors Microcontroller Data Handbook (IC20)).
– Load MyAddr RAM location with the address of this node.
– For Slave operation, load STxBuf with the initial data to be transmitted.
– For slave operation, set the SLAVEN bit in the I2CFG register.
– Enable I^{2}C and watchdog interrupts by setting the ETI, EI2 and EA bits of the interrupt enable register.
– For Master operation, set up the next transaction by loading the appropriate directives into MASCMD, DESTADRW, DESSUBAD (if applicable) and MASTCNT, and load MasBuf with the appropriate data if it is a Write message.
– For Master operation, initiate the next transaction by setting MASTRQ bit in I2CFG.
– For both Master and Slave operation, handle data transmission and reception via the buffers in main-line code or the Event Routines.

PROGRAMMING EXAMPLE

The assembler listing includes the I^{2}C Communications Routines and a demo application exercising these routines. In most real-life applications the code of the routines could be used without modifications. For those who follow the coding of the routines, one should note that in many instances code speed and program space have been slightly compromised in order to improve readability. The almost "general purpose" interface to the routines affects efficiency as well, and it is possible to write more compact and somewhat faster code for specific applications. The reader is encouraged, though, to use the code "as is" whenever possible.

The "application" demo is simple—two microcontrollers exchange messages in a "ping-pong" game. In addition to trivial message exchange, the code demonstrates recovery mechanisms from communications errors and bus "hangups". We tried this code with two pairs of controllers exchanging messages on the same bus. The message exchange could repeatedly recover and restart when the SCL and SDA lines were temporarily shorted to ground or between themselves. Simpler versions, without the "protection" mechanisms, could "hang up" under such conditions.

Source Code Available On BBS

The source code file for this program is available for download 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.

```
PPCODE1     83C751 Multimaster I2C Routines                 4/14/1992      PAGE 1

            ;

            ;****************************************************************************
            ;               Multimaster Code for 83C751/83C752
            ;               4/14/1992
            ;****************************************************************************
            ; This code was written to accompany an application note. The I2C routines
            ; are intended to be demonstrative and transportable into different
            ; application scenarios, and were NOT optimized for speed and/or memory
            ; utilization.
            ;
            ; Yoram Arbel

            $TITLE(83C751 Multi Master I2C Routines)
            $DATE(4/14/1992)
            $MOD751
            $DEBUG

            ;****************************************************************************
            ;               8XC751 MULTIMASTER I2C COMMUNICATIONS ROUTINES
            ;               Symbols and RAM definitions
            ;****************************************************************************

            ; Symbols (masks) for I2CFG bits.

0010        BTIR        EQU     10h     ; TIRUN bit.
0040        BMRQ        EQU     40h     ; MASTRQ bit.


            ; Symbols (masks) for I2CON bits.

0080        BCXA        EQU     80h     ; CXA bit.
0040        BIDLE       EQU     40h     ; IDLE bit.
0020        BCDR        EQU     20h     ; CDR bit.
0010        BCARL       EQU     10h     ; CARL bit.
0008        BCSTR       EQU     08h     ; CSTR bit.
0004        BCSTP       EQU     04h     ; CSTP bit.
0002        BXSTR       EQU     02h     ; XSTR bit.
0001        BXSTP       EQU     01h     ; XSTP bit.

            ; Note:
            ;
            ; Specific bits of the I2CON register are set by writing into this register a
            ; combination of the masks defined above using the MOV command.
            ; The SETB command should not be used with I2CON, as it is implemented by
            ; reading the contents of the register, setting the appropriate bit and
            ; writing it back into the register. As the functionality of the Read and
            ; Write portions of the I2CON register is different, using SETB may cause
            ; unwanted results.

            ; Message transaction status indications in MSGSTAT:

0010        SGO         EQU     10h     ; Started Slave message processing.
```

```
PPCODE1         83C751 Multimaster I2C Routines                       4/14/1992     PAGE 2

0011       SRCVD       EQU     11h     ; as a slave, received a new message
0012       SRLNG       EQU     12h     ; received as slave a message which is too
                                       ; long for the buffer
0013       STXED       EQU     13h     ; as slave, completed message transmission.
0014       SRERR       EQU     14h     ; bus error detected when operating as a slave.

0020       MGO         EQU     20h     ; Started Master message processing.
0021       MRCVED      EQU     21h     ; As Master, received complete message from
                                       ; slave.
0022       MTXED       EQU     22h     ; As Master, completed successful message
                                       ; transmission (slave acknowledged all data
                                       ; bytes).
0023       MTXNAK      EQU     23h     ; As Master, truncated message since slave did
                                       ; not acknowledge a data byte.
0024       MTXNOSLV    EQU     24h     ; AS Master, did not receive an acknowledgement
                                       ; for the specified slave address.

0030       TIMOUT      EQU     30h     ; TIMERI Timed out.
0032       NOTSTR      EQU     32h     ; Master did not recognize Start.

                                       ; RAM locations used by I2C interrupt service routines.


0020       MASCMD      DATA    20h
0000       SUBADD      BIT     MASCMD.0
0001       RPSTRT      BIT     MASCMD.1
0002       SETMRQ      BIT     MASCMD.2

0024       DSEG        AT      24h

0024       MSGSTAT:    DS      1       ; I2C communications status.
0025       MYADDR:     DS      1       ; Address of this I2C node.
0026       DESTADRW:   DS      1       ; Destination address + R/W (for Master).
0027       DESSUBAD:   DS      1       ; Destination subaddress.
0028       MASTCNT:    DS      1       ; Number of data bytes in message (Master,
                                       ; send or receive).

0029       TITOCNT:    DS      1       ; Timer I bus watchdog timeouts counter.
002A       StackSave:  DS      1       ; SP save location (used when returning from
                                       ; bus recovery routine).

002B       MasBuf:     DS      4       ; Master receive/transmit buffer, 8 bytes.
002F       SRcvBuf:    DS      4       ; Slave receive buffer, 8 bytes.
0033       STxBuf:     DS      4       ; Slave transmit buffer, 8 bytes.



0004       RBufLen     EQU     4h      ; The length of SRcvBuf

```

```
PPCODE1          83C751 Multimaster I2C Routines                   4/14/1992      PAGE 3

                 ;*****************************************************************
                 ;                APPLICATION output pins and RAM definitions
                 ;*****************************************************************

                 ; Outputs used by the application:

0090             TogLED       BIT       P1.0       ; Toggling output pin, to confirm
                                                   ; that the ping–pong game proceeds fine.
0091             ErrLED       BIT       P1.1       ; Error indication.

0093             OnLED        BIT       P1.3       ;

                 ; Application RAM

0021             APPFLAGS     DATA      21h
0008             TRQFLAG      BIT       APPFLAGS.0
                 ; Flag for monitoring I2C transmission success.
0009             SErrFLAG     BIT       APPFLAGS.1

0037             FAILCNT:     DS        1

0038             TOGCNT:      DS        1          ; Toggle counter.


                 ;*****************************************************************
                 ;
                 ;                Program Start
                 ;
                 ;*****************************************************************
——               CSEG

                 ; Reset and interrupt vectors.

0000 4178        AJMP         Reset                ;Reset vector at address 0.


                 ; A timer I timeout usually indicates a ’hung’ bus.

001B             ORG          1Bh                  ; Timer I (I2C timeout) interrupt.
001B D2DD        TimerI:      SETB      CLRTI
001D 4111        AJMP         TIISR                ; Go to Interrupt Service Routine.




                 ;*************************************************************************
                 ;                I2C Interrupt Service Routine
                 ;*************************************************************************
                 ;
                 ; Notes on the interrupt mechanism:
                 ;
                 ; Other interrupts are enabled during this ISR upon return from XRETI.
                 ; Limitations imposed on other ISR’s:
```

```
PPCODE1      83C751 Multimaster I2C Routines                       4/14/1992       PAGE 4

                     ; – Should not be long (close to 1000 clock cycles).        A long ISR will cause
                     ;the I2C bus to ’hang”, and a TIMERI interrupt to occur.
                     ; – Other interrupts either do not use the same mechanism for allowing
                     ;further interrupts, or if they do – disable TIMERI interrupt beforehand.
                     ;
                     ; The 751 hardware allows only one level of interrupts. We simulate an
                     ; additional level by software: by performing a RETI instruction (at location
                     ; XRETI) the interrupt–in–progress flip–flop is cleared, and other interrupts
                     ; are enabled.     The second level of interrupt is a must in our implementation,
                     ; enabling timeout interrupts to occur during ”stuck” wait loops in the I2C
                     ; interrupt service routine.


0023                 ORG          23h

0023 C2AC            I2CISR:      CLR        EI2        ; Disable I2C interrupt.
0025 114C            ACALL        XRETI                 ; Allow other interrupts to occur.
0027 C0D0            PUSH         PSW
0029 C0E0            PUSH         ACC
002B E8              MOV          A,R0
002C C0E0            PUSH         ACC
002E E9              MOV          A,R1
002F C0E0            PUSH         ACC
0031 EA              MOV          A,R2
0032 C0E0            PUSH         ACC

0034 85812A          MOV          StackSave, SP
0037 C2DC            CLR          TIRUN
0039 D2DC            SETB         TIRUN

003B 209A09          JB           STP,NoGo
003E 30990C          JNB          MASTER, GoSlave
0041 752420          MOV          MSGSTAT,#MGO
0044 209B76          JB           STR,GoMaster
0047 752432          NoGo:        MOV        MSGSTAT,#NOTSTR
004A 21AE            AJMP         Dismiss               ; Not a valid Start.

004C 32              XRETI:       RETI

                     ;****************************************************************************
                     ;            Main Transmit and Receive Routines
                     ;****************************************************************************

                     ;            SLAVE CODE –
                     ;            GET THE ADDRESS

004D 752410          GoSlave:     MOV        MSGSTAT,#SGO
0050 31E2            AddrRcv:     ACALL      ClsRcv8
0052 309D5E          JNB          DRDY, SMsgEnd         ; Must be some strange Start or Stop
                                                        ; before the address byte was completed.
                                                        ; Not a valid address.
0055 A2E0            STstRW:      MOV        C,ACC.0    ; Save R/W~ bit in carry.
0057 C2E0            CLR          ACC.0                 ; Clear that bit, leaving ”raw” address
```

```
PPCODE1         83C751 Multimaster I2C Routines                         4/14/1992         PAGE 5

0059 6060               JZ          GoIdle      ; If it is a General Address
                                                ; – ignore it.

                                                ; NOTE:
                                                ; One may insert here a different
                                                ; treatment for general calls, if
                                                ; these are relevant.

005B 4027               JC          SlvTx       ; It's a Read – (requesting slave
                                                ; transmit).




                                                ; It is a Write (slave should receive the message).

                                                ; Check if message is for us

005D B5255B             SRcv2:      CJNE        A,MYADDR,GoIdle        ; If not my address – ignore the
                                                                       ; message.
0060 792F               MOV         R1,#SRcvBuf         ; Set receive buffer address.
0062 7A05               MOV         R2,#RbufLen+1       ;
0064 8002               SJMP        SRcv3

0066 F7                 SRcvSto:    MOV         @R1,A   ; Store the byte
0067 09                 Inc         R1                  ; Step address.
0068 31ED               SRcv3:      ACALL       AckRcv8
006A 309D09             JNB         DRDY,SRcvEnd        ; Exit loop –end reception.
006D DAF7               DJNZ        R2,SRcvSto          ; Go to store byte if buffer not full.

                                                        ; Too many bytes received – do not acknowledge.
006F 752412             MOV         MSGSTAT,#SRLNG      ; Notify main that (as slave) we
                                                        ; have received too long a message.
0072 7110               ACALL       SLnRCvdR            ; Handle new data – slave event routine.
0074 8045               SJMP        GoIdle



                        ; Received a byte, but not DRDY – check if a legitimate message end.

0076 B8072E             SRcvEnd:    CJNE        R0,#7,SRcvErr       ; If bit count not 7, it was not
                                                                    ; a Start or a Stop.

                                                                    ; Received a complete message


0079 752411             MOV         MSGSTAT,#SRCVD
                                                                    ; Calculate number of bytes received
007C E9                 MOV         A,R1
007D C3                 CLR         C
007E 942F               SUBB        A,#SRcvBuf                      ; number of bytes in ACC
0080 51EF               ACALL       SRCvdR                          ; Handle new data – slave event routine.
0082 802F               SJMP        SMsgEnd
```

```
PPCODE1      83C751 Multimaster I2C Routines                         4/14/1992     PAGE 6



                                                 ; It is a Read message, check if for us.

0084 00        SlvTx:      NOP

0085 B52533    STx2:       CJNE    A,MYADDR,GoIdle ; Not for us.
0088 759900    MOV         I2DAT,#0              ; Acknowledge the address.
008B 309EFD    JNB         ATN,$                 ; Wait for attention flag.
008E 309D22    JNB         DRDY,SMsgEnd          ; Exception – unexpected Start
                                                 ; or Stop before the Ack got out.
0091 7933      MOV         R1,#STxBuf            ; Start address of transmit buffer.
0093 E7        STxlp:      MOV     A,@R1         ; Get byte from buffer
0094 09        INC         R1
0095 31CE      ACALL       XmByte
0097 309D19    JNB         DRDY,SMsgEnd          ; Byte Tx not completed.
009A 309FF6    JNB         RDAT,STxlp            ; Byte acknowledge, proceed trans.
009D 759860    MOV         I2CON,#BCDR+BIDLE     ; Master Nak'ed for msg end.
00A0 752413    MOV         MSGSTAT,#STXED
00A3 7110      ACALL       STXedR                ; Slave transmitted event routine.
00A5 21AE      AJMP        Dismiss


00A7 752414    SRcvErr:    MOV     MSGSTAT,#SRERR        ; Flag bus/protocol error
00AA 7110      ACALL       SRErrR                        ; Slave error event routine.
00AC 8005      SJMP        SMsgEnd
00AE 752414    StxErr:     MOV     MSGSTAT,#SRERR        ; Flag bus/protocol error
00B1 7110      ACALL       SRErrR

00B3 209903    SMsgEnd:    JB      MASTER,SMsgEnd2
00B6 209B94    JB          STR,GoSlave                   ; If it was a Start, be Slave
00B9           SMsgEnd2:
00B9 21AE      AJMP        Dismiss


                                                         ; End of Slave message processing

00BB           GoIdle:
00BB 21AE      AJMP        Dismiss




               ;
               ;

00BD           GoMaster:


               ; Send address & R/W~ byte

00BD 792B      MOV         R1,#MasBuf            ; Master buffer address
00BF AA28      MOV         R2,MASTCNT            ; # of bytes, to send or rcv
```

```
PPCODE1     83C751 Multimaster I2C Routines                           4/14/1992       PAGE 7

00C1 E526        MOV        A,DESTADRW                 ; Destination address (including
                                                       ; R/W~ byte).
00C3 200012      JB         SUBADD,GoMas2              ; Branch if subaddress is needed.

00C6 31C5        ACALL      XmAddr

00C8 309D03      JNB        DRDY,GM2
00CB 309C02      JNB        ARL,GM3
00CE 2186        GM2:       AJMP     AdTxArl           ; Arbitration loss while transmitting
                                                       ; the address.
00D0 209F5C      GM3:       JB       RDAT,Noslave      ; No Ack for address transmission.
00D3 20E063      JB         ACC.0, MRcv                ; Check R/W~ bit
00D6 211A        AJMP       MTx

                                                       ; Handling subaddress case:

00D8 00          GoMas2:    NOP                        ; Subaddress needed. Address in ACC.
00D9 C2E0        CLR        ACC.0                      ; Force a Write bit with address.
00DB 31C5        ACALL      XmAddr
00DD 309D03      JNB        DRDY,GM4
00E0 309C02      JNB        ARL,GM5
00E3 2186        GM4:       AJMP     AdTxArl           ; Arbitration loss while transmitting
                            ; the address.

00E5 209F47      GM5:       JB       RDAT,Noslave      ; No Ack for address transmission.
00E8 E527        MOV        A,DESSUBAD
00EA 31CE        ACALL      XmByte                     ; Transmit subaddress.
00EC 309DCA      JNB        DRDY,SMsgEnd2              ; Arbitration loss (by Start or Stop)
00EF 209CC7      JB         ARL,SMsgEnd2               ; Arbitration loss occurred.
00F2 209F3F      JB         RDAT,NoAck                 ; Subaddress transmission was not ack'ed.
00F5 E526        MOV        A,DESTADRW                 ; Reload ACC with address.
00F7 30E020      JNB        ACC.0, MTx                 ; It's a Write, so proceed
                            ; by sending the data.
                                                       ; Read message, needs rp. Start and add. retransmit.

00FA 759822      MOV        I2CON,#BCDR+BXSTR          ; Send Repeated Start.
00FD 309EFD      JNB        ATN,$
0100 759820      MOV        I2CON,#BCDR                ; Clear useless DRDY while preparing
                                                       ; for Repeated Start.
0103 309EFD      JNB        ATN,$                      ; expecting an STR.
0106 309C02      JNB        ARL,GM6
0109 2182        AJMP       MArlEnd                    ; oops – lost arbitration.
010B 31C5        GM6:       ACALL    XmAddr            ; Retransmit address, this time with the
                                                       ; Read bit set.
010D 309D03      JNB        DRDY,GM7
0110 309C02      JNB        ARL,GM8
0113 2186        GM7:       AJMP     AdTxArl           ; Arbitration loss while transmitting
                                                       ; the address.
0115 209F17      GM8:       JB       RDAT,Noslave      ; No Ack – the slave disappeared.
0118 801F        SJMP       MRcv                       ; Proceed receiving slave's data.

                 ; A Write message.    Master transmits the data.

```

```
PPCODE1        83C751 Multimaster I2C Routines                              4/14/1992      PAGE 8

011A 00        MTx:         NOP

011B E7        MTxLoop:     MOV      A,@R1              ; Get byte from buffer.
011C 09        INC          R1                          ; Step the address.
011D 31CE      ACALL        XmByte
011F 309D97    JNB          DRDY,SMsgEnd2               ; Arbitration loss (by Start or Stop)
0122 209C94    JB           ARL,SMsgEnd2                ; Arbitration loss.
0125 209F0C    JB           RDAT,NoAck
0128 DAF1      DJNZ         R2,MTxLoop                  ; Loop if more bytes to send.

012A 752422    MOV          MSGSTAT,#MTXED              ; Report completion of buffer
                                                        ; transmission.
012D 8025      SJMP         MTxStop
012F 752424    NoSlave:     MOV      MSGSTAT,#MTXNOSLV
0132 8020      SJMP         MTxStop
0134 752423    NoAck:       MOV      MSGSTAT,#MTXNAK
0137 801B      SJMP         MTxStop



               ; Master receive – a Read frame

0139 31F6      MRcv:        ACALL    ClaRcv8            ; Receive a byte.
013B 8002      SJMP         MRcv2
013D 31ED      MRcvLoop:    ACALL    AckRcv8
013F 309D39    MRcv2:       JNB      DRDY,MArl          ; Other’s Start or Stop.
0142 F7        MOV          @R1,A                       ; Store received byte.
0143 09        INC          R1                          ; Advance address.
0144 DAF7      DJNZ         R2,MRcvLoop

               ; Received the desired number of bytes – send Nack.

0146 759980    MOV          I2DAT,#80h
0149 309EFD    JNB          ATN,$
014C 309D2C    JNB          DRDY,MArl
014F 752421    MOV          MSGSTAT,#MRCVED
0152 8000      SJMP         MTxStop                     ; Go to send Stop or Repeated Start.



               ; Conclude this Master message:
               ; Send Stop, or a Repeated Start


0154 300105    MTxStop:     JNB      RPSTRT,MTxStop2            ; Check if Repeated Start needed
                                                                ; Around if not RPSTRT.
0157 759822    MOV          I2CON,#BCDR+BXSTR                   ; Send Repeated Start.
015A 8007      SJMP         MTxStop3
015C A202      MTxStop2:    MOV      C,SETMRQ                   ; Set new Master Request if demanded
015E 92DE      MOV          MASTRQ,C                            ; by SETMRQ bit of MASCMD.

0160 759821    MOV          I2CON,#BCDR+BXSTP                   ; Request the HW to send a Stop.

```

```
PPCODE1         83C751 Multimaster I2C Routines                                 4/14/1992       PAGE 9

0163 309EFD     MTxStop3:       JNB      ATN,$               ; Wait for Attention
0166 759820     MOV             I2CON,#BCDR                  ; Clear the useless DRDY, generated
                                                             ; by SCL going high in preparation
                                                             ; for thr Stop or Repeated Start.
0169 309EFD     JNB             ATN,$                        ; Wait for ARL, STP or STR.
016C 209C13     JB              ARL,MarlEnd                  ; Lost arbitration trying to send
                                                             ; Stop or a ReStart.

                ; Master is done with this message.    May proceed with new messages, if any,
                ; or exit.

016F 7112       ACALL           MastNext                     ; Master Event Routine. May Prepare
                                                             ; the pointers and data for the
                                ;                             next Master message.

0171 30DE05     JNB             MASTRQ,MMsgEnd               ; Go end service routine if MASTRQ
                                                             ; does not indicate that the master
                                                             ; should continue (was set according
                                                             ; to SETMRQ bit, or by MastNext).

0174 309B02     JNB             STR,MMsgEnd                  ; Return from the ISR, unless Start
                                                             ; (avoid danger if we do not return:
                                                             ; if there was a Stop, the watchdog
                                                             ; is inactive until next Start).
0177 01BD       AJMP            GoMaster                     ; Loop for another Master message
                                ;
0179            MMsgEnd:                                     ; End of Master messages,
0179 8033       SJMP            Dismiss





                ; Terminate mastership due to an arbitration loss:

017B            MArl:

017B 309B02     JNB             STR,MArl2                    ; If lost arbitration due to other
                                                             ; Master's Start, go be a slave.
017E 014D       AJMP            GoSlave

0180            Marl2:
0180 21AE       AJMP            Dismiss




                ; Switch from Master to Slave due to arbitration loss after completing
                ; transmission of a message.   The MASTRQ bit was cleared trying to write a
                ; Stop, and we need to set it again on order to retry transmission when the
                ; bus gets free again.

0182            MArlEnd:
0182 D2DE       SETB            MASTRQ                       ; Set Master Request – which will get
                                                             ; into effect when we are done as a
                                                             ; slave.
```

```
PPCODE1          83C751 Multimaster I2C Routines                          4/14/1992        PAGE 10

0184 217B                AJMP        MArl

                         ; Handling arbitration loss while transmitting an address

0186 209BF2              AdTxArl:    JB       STR,MArl          ; Non–synchronous Start or Stop.
0189 209AEF              JB          STP,MArl

                         ; Switch from Master to Slave due to arbitration loss while transmitting
                         ; an address – complete receiving the address transmitted by the new Master.

018C B80003              CJNE        R0,#0,AdTxArl2
                                                                ; Arl on last bit of address
                                                                ; (R0 is 0 on exit from XmAddr).
018F 14                  DEC         A                          ; The lsb sent, in which arl occured
                                                                ; must have been 1. By decrementing
                                                                ; A we get the address that won.
0190 8012                SJMP        AdAr3

0192                     AdTxArl2:
0192 03                  RR          A                          ; Realign partially Tx’ed ACC
0193 F9                  MOV         R1,A                       ; and save it in R1
0194 E8                  MOV         A,R0                       ; Pointer for lookup table
0195 9001A6              MOV         DPTR,#MaskTable
0198 93                  MOVC        A,@A+DPTR
0199 59                  ANL         A,R1                       ; Set address bits to be received,
                                                                ; and the bit on which we lost
                                                                ; arbitration to 0
                                                                ; Now we are ready to receive the rest
                                                                ; of the address.


019A 759890              MOV         I2CON,#BCXA+BCARL          ; Clear flags and release the clock.

019D 5108                ACALL       RBit3                      ; Complete the address using reception
                                                                ; subroutine.
019F 209D02              JB          DRDY,AdAr3                 ; Around if received address OK
01A2 01B3                AJMP        SMsgEnd                    ; Unexpected Start or Stop – end
                                                                ; as a slave.
01A4 0155                AdAr3:      AJMP     STstRW            ; Proceed to check the address
                                                                ; as a slave.

01A6 FF7E3E1E            MaskTable:  DB       0ffh,7Eh,3Eh,1Eh,0Eh,06h,02h,00h, ; 0ffh is dummy
01AA 0E060200

                         ; End I2C Interrupt Service Routine:

01AE 711E                Dismiss:    ACALL    I2CDONE

01B0 7598F4              MOV         I2CON,#BCARL+BCSTP+BCDR+BCXA+BIDLE
01B3 C2DC                CLR         TIRUN
01B5 D0E0                POP         ACC
01B7 FA                  MOV         R2,A
01B8 D0E0                POP         ACC
```

```
PPCODE1          83C751 Multimaster I2C Routines                           4/14/1992         PAGE 11

01BA F9               MOV           R1,A
01BB D0E0             POP           ACC
01BD F8               MOV           R0,A
01BE D0E0             POP           ACC
01C0 D0D0             POP           PSW
01C2 D2AC             SETB          EI2

01C4 22               RET           ; Return from I2C interrupt Service Routine

                      ;****************************************************************************
                      ;             Byte Transmit and Receive Subroutines
                      ;****************************************************************************



                      ;             XmAddr: Transmit Address and R/W~
                      ;             XmByte: Transmit a byte

01C5 F599             XmAddr:       MOV      I2DAT,A                   ; Send first bit, clears DRDY.
01C7 75981C           MOV           I2CON,#BCARL+BCSTR+BCSTP
                                                                       ; Clear status, release SCL.
01CA 7808             MOV           R0,#8                              ; Set R0 as bit counter
01CC 8004             SJMP          XmBit2
01CE 7808             XmByte:       MOV      R0,#8
01D0 F599             XmBit:        MOV      I2DAT,A                   ; Send the first bit.
01D2 23               XmBit2:       RL       A                         ; Get next bit.
01D3 309EFD           JNB           ATN,$                              ; Wait for bit sent.
01D6 309D08           JNB           DRDY,XmBex                         ; Should be data ready.
01D9 D8F5             DJNZ          R0,XmBit                           ; Repeat until all bits sent.
01DB 7598A0           MOV           I2CON,#BCDR+BCXA                   ; Switch to receive mode.
01DE 309EFD           JNB           ATN,$                              ; Wait for acknowledge bit.
                                                                       ; flag cleared.
01E1 22               XmBex:        RET

                      ;
                      ; Byte receive routines.
                      ;
                      ; ClsRcv8       clears the status register (from Start condition)
                      ;               and then receives a byte.
                      ; AckRcv8       Sends an acknowledge, and then receives a new byte.
                      ;               If a Start or Stop is encountered immediately after the
                      ;               ack, AckRcv8 returns with 7 in R0.
                      ; ClaRcv8       clears the transmit active state and releases clock
                      ;               (from the acknowledge).
                      ;
                      ;               A contains the received byte upon return.
                      ;               R0 is being used as a bit counter.
                      ;

01E2 75989C           ClsRcv8:      MOV      I2CON,#BCARL+BCSTR+BCSTP+BCXA
                      ;Clear status register.
01E5 309EFD           JNB           ATN,$
01E8 309D22           JNB           DRDY,RCVex
```

```
PPCODE1        83C751 Multimaster I2C Routines                          4/14/1992     PAGE 12

01EB 800F         SJMP        Rcv8

01ED 759900       AckRcv8:    MOV     I2DAT,#0                ; Send Ack (low)
01F0 309EFD       JNB         ATN,$
01F3 309D18       JNB         DRDY,RCVerr                     ; Bus exception – exit.
01F6 7598A0       ClaRcv8:    MOV     I2CON,#BCDR+BCXA        ; clear status, release clock
                                                              ;from writing the Ack.
01F9 309EFD       JNB         ATN,$

01FC 7807         Rcv8:       MOV     R0,#7                   ; Set bit counter for the first seven
                                                              ; bits.
01FE E4           CLR         A                               ; Init received byte to 0.
01FF 4599         RBit:       ORL     A,I2DAT                 ; Get bit, clear ATN.
0201 23           RBit2:      RL      A                       ; Shift data.
0202 309EFD       JNB         ATN,$                           ; Wait for next bit.
0205 309D05       JNB         DRDY,RCVex                      ; Exit if not a data bit (could be Start/
                                                              ; Stop, or bus/protocol error)
0208 D8F5         RBit3:      DJNZ    R0,RBit                 ; Repeat until 7 bits are in.
020A A29F         MOV         C,RDAT                          ; Get last bit, don't clear ATN.
020C 33           RLC         A                               ; Form full data byte.
020D 22           RCVex:      RET

020E 7809         RCVerr:     MOV     R0,#9                   ; Return non legitimate bit count
0210 22           RET



                  ;*****************************************************************************
                  ;           Timer I Interrupt Service Routine
                  ;           I2C us Timeout
                  ;*****************************************************************************

                  ; In addition to reporting the timeout in MSGSTAT, we update a failure
                  ; counter, TITOCNT. This allows different types of timeout handling by the
                  ; main program.

0211 C2DE         TIISR:      CLR     MASTRQ                  ; "Manual" reset.
0213 759801       MOV         I2CON,#BXSTP                    ;
0216 7598BC       MOV         I2CON,#BCXA+BCDR+BCARL+BCSTR+BCSTP

0219 752430       TI1:        MOV     MSGSTAT,#TIMOUT         ; Status Flag for Main.
021C 74FF         TI2:        MOV     A,#0FFh
021E B52902       CJNE        A,TITOCNT,TI3                   ; Increment TITOCNT, saturating
0221 8002         SJMP        TI4                             ; at FFh.
0223 0529         TI3:        INC     TITOCNT

0225 5130         TI4:        ACALL   RECOVER

0227 D2DD         SETB        CLRTI                           ; Clear TI interrupt flag.
0229 114C         ACALL       XRETI                           ; Clear interrupt pending flag (in
                                                              ; order to re–enable interrupts).
022B 852A81       MOV         SP,StackSave                    ; Realign stack pointer, re–doing
                                                              ; possible stack changes during
                                                              ; the I2C interrupt service routine.
```

```
PPCODE1        83C751 Multimaster I2C Routines                                4/14/1992       PAGE 13

                                           ; TimerI interrupts in other ISR's
                                           ; were not allowed !
022E 21AE          AJMP                    Dismiss      ; Go back to the I2C service routine,
                                           ; in order to return to the (main)
                                           ; program interrupted.


                   ;*******************************************************************************
                   ;                       Bus recovery attempt subroutine
                   ;*******************************************************************************

0230 C2AF          RECOVER:                CLR       EA
0232 C2DE          CLR                     MASTRQ                            ; "Manual" reset.
0234 7598FC        MOV                     I2CON,#BCXA+BIDLE+BCDR+BCARL+BCSTR+BCSTP
0237 C2DF          CLR                     SLAVEN                            ; Non I2C TimerI mode
0239 D2DC          SETB                    TIRUN                             ; Fire up TimerI. When it overflows, it
                                                                             ; will cause I2C interface hardware reset.
023B 79FF          MOV                     R1,#0ffh
023D 00            DLY5:                   NOP
023E 00            NOP
023F 00            NOP
0240 D9FB          DJNZ                    R1,DLY5
0242 C2DC          CLR                     TIRUN
0244 D2DD          SETB                    CLRTI

0246 D280          SETB                    SCL                               ; Issue clocks to help release other devices.
0248 D281          SETB                    SDA
024A 7908          MOV                     R1,#08h
024C C280          RC7:                    CLR       SCL
024E 00000000      DB                      0,0,0,0,0
0252 00
0253 D280          SETB                    SCL
0255 00000000      DB                      0,0,0,0,0
0259 00
025A D9F0          DJNZ                    R1,RC7
025C C280          CLR                     SCL
025E 0000          DB                      0,0
0260 C281          CLR                     SDA
0262 0000          DB                      0,0
0264 D280          SETB                    SCL
0266 00000000      DB                      0,0,0,0,0
026A 00
026B D281          SETB                    SDA
026D 00000000      DB                      0,0,0,0,0                         ; Issue a Stop.
0271 00

0272 7598BC        Rex:                    MOV       I2CON,#BCXA+BCDR+BCARL+BCSTR+BCSTP ; clear flags
0275 D2AF          SETB                    EA
0277 22            RET

```

```
PPCODE1      83C751 Multimaster I2C Routines                    4/14/1992      PAGE 14

             ;***************************************************************************
             ;
             ;                Main Program
             ;
             ;***************************************************************************

             ; Message ping pong game. Each message is transmitted by
             ; a processor that is a master on the I2C bus, and it contains one byte
             ; of data. A processor that receives this data byte as a slave increments
             ; the data by one and transmits it back as a master. The data received is
             ; confirmed to be a one increment of the data formerly sent, unless
             ; it is a ”reset” value, chosen to be 00h.
             ; The two participating processors have similar code, where the node
             ; address of the second processor is the destination address of this
             ; one, and vice versa.
             ; The first data byte each processor tries to send is 00h. One of the
             ; processors will acquire the bus first, and the second processor that will
             ; receive this ”resetting” 00h will not attempt to confirm it against an
             ; expected value.  It will simply increment and transmit it. Subsequent
             ; receptions will be confirmed against the expected value, until 0ffh data
             ; bytes are sent and the game is effectively reset by the 00h resulting from
             ; the next increment.
             ; A toggling output (TogLED) tells the outer world that the ”ping pong”
             ; proceeds well. If something unexpected happens we temporarily activate
             ; another output, ErrLED.
             ; The different tasks of the code are performed in a combination of main–
             ; line program and event routines called from the I2C interrupt service
             ; routine.


             ; Initial set–ups:
             ; Load CT1,CT0 bits of I2CFG register, according to the clock
             ; crystal used.
             ; Load RAM location MYADDR with the I2C address of this processor.
             ; We load these values out of ROM table locations (R_CTVAL and R_MYADDR).
             ; One may, instead, load with a MOV <immediate> command.

0278 758107  Reset:          MOV       SP,#07h   ;Set stack location.
027B E4      CLR             A
027C 90032D  MOV             DPTR,#R_CTVAL
027F 93      MOVC            A,@A+DPTR
0280 F5D8    MOV             I2CFG,A             ; Load CT1,CT0 (I2C timing, crystal
                                                 ; dependent).
0282 E4      CLR             A
0283 90032C  MOV             DPTR,#R_MYADDR
0286 93      MOVC            A,@A+DPTR           ; Get this node’s address from ROM table
0287 F525    MOV             MYADDR,A            ; into MYADDR RAM location.

0289 C293    CLR             OnLED


028B C291    Reset2:         CLR       ErrLED    ; Flash LED.
028D 51E6    ACALL           LDELAY
```

```
PPCODE1        83C751 Multimaster I2C Routines                4/14/1992      PAGE 15

028F D291      SETB      ErrLED
0291 C209      CLR       SErrFLAG
0293 C208      CLR       TRQFLAG
0295 753750    MOV       FAILCNT,#50h
0298 D290      SETB      TogLED
029A 753850    MOV       TOGCNT,#050h        ; Initialize pin–toggling counter

               ; Enable slave operation.
               ; The Idle bit is set here for a restart situation – in normal
               ; operation this is redundant, as this bit is set upon power_up reset.
029D 759840    MOV       I2CON,#BIDLE        ; Slave will idle till next Start.
02A0 D2DF      SETB      SLAVEN              ; Enable slave operation.

               ; Enable interrupts.
               ; This is necessary for both Slave and Master operations.
02A2 D2AB      SETB      ETI                 ; Enable timer I interrupts.
02A4 D2AC      SETB      EI2                 ; Enable I2C port interrupts.
02A6 D2AF      SETB      EA                  ; Enable global interrupts.

               ; Set up Master operation.

02A8 752000    MOV       MASCMD,#0h          ; ”Regular” master transmissions.
02AB 90032E    MOV       DPTR,#PongADDR
02AE E4        CLR       A
02AF 93        MOVC      A,@A+DPTR
02B0 F526      MOV       DESTADRW,A          ; The partner address. The LSB is
                                             ; low, for a Write transaction.
02B2 752801    MOV       MASTCNT,#01h        ; Message length – a single byte.

02B5           PPSTART:
02B5 752B00    MOV       MasBuf,#00h

               ; ”Ping” transmission:

02B8           PP2:
02B8 D208      SETB      TRQFLAG
02BA D2DE      SETB      MASTRQ
02BC 79FF      MOV       R1,#0ffh
02BE 300809    PP22:     JNB      TRQFLAG,PP3      ; Transmitted OK
02C1 D9FB      DJNZ      R1,PP22
02C3 D537F2    MFAIL1:   DJNZ     FAILCNT,PP2
02C6 5130      ACALL     RECOVER
02C8 80C1      SJMP      Reset2

               ; ”Pong” reception:

02CA 78FF      PP3:      MOV      R0,#0ffh         ; Software timeout loop count.
02CC 79FF      PP31:     MOV      R1,#0ffh
02CE 2008E7    PP32:     JB       TRQFLAG,PP2      ; Rcvd ok as slave, go transmit.
02D1 200908    JB        SErrFLAG,PP5
02D4 D9F8      DJNZ      R1,PP32
02D6 D8F4      DJNZ      R0,PP31
02D8 5130      PPTO:     ACALL    RECOVER          ; Software timeout.
```

```
PPCODE1        83C751 Multimaster I2C Routines                          4/14/1992      PAGE 16

02DA 418B          AJMP        Reset2

02DC C291          PP5:        CLR      ErrLED   ; Receive error.
02DE 51E6          ACALL       LDELAY
02E0 D291          SETB        ErrLED
02E2 C209          CLR         SErrFLAG
02E4 41B5          AJMP        PPSTART

02E6 7A30          LDELAY:     MOV      R2,#030h
02E8 79FF          LDELAY1:    MOV      R1,#0ffh
02EA D9FE          DJNZ        R1,$
02EC DAFA          DJNZ        R2,LDELAY1
02EE 22            RET

                   ;*****************************************************************************
                   ;           Slave and Master Event Routines.
                   ;*****************************************************************************

                   ;
                   ;Invoked upon completion of a message transaction.
                   ;This is the part of the application program actually dealing
                   ;with the data communicated on the I2C bus, by responding to
                   ;new data received and/or preparing the next transaction.


                   ; Slave Event Routines
                   ;
                   ; These routines are invoked by the I2C interrupt service routine when a
                   ; message transaction as a slave has been completed. Our ”application”
                   ; reacts to a message received as a slave with the routine SRCvdR.
                   ; The calls that indicate erroneous reception are treated the same way as
                   ; erroneous data reception in the ”ping pong” game.

                   ;SRcvdR
                   ;Invoked when a new message has been received as a Slave.

02EF 00            SRcvdR:     NOP
02F0 E52F          MOV         A,SRcvBuf
02F2 7005          JNZ         SR2
02F4 752B01        MOV         MasBuf,#01h       ; It was ping–pong reset value
02F7 800F          SJMP        SR3

02F9 052B          SR2:        INC      MasBuf   ; The expected data.
02FB B52B0F        CJNE        A,MasBuf,ErrSR
02FE 052B          INC         MasBuf            ; Data for next transmission – the data
                                                 ; received incremented by 1.

                   ;A successful two way data exchange.  Let the outside world know by
                   ;toggling an output pin driving a LED.  We actually toggle only
                   ;when a number of such exchanges is completed, in order to
                   ;slow down the changes for a good visual indication.

```

```
PPCODE1         83C751 Multimaster I2C Routines                           4/14/1992      PAGE 17

0300 D53805     DJNZ         TOGCNT,SR3
0303 B290       CPL          TogLED              ; Toggle output
0305 753850     MOV          TOGCNT,#050h        ;

0308 C209       SR3:         CLR       SErrFLAG
030A D208       SETB         TRQFLAG             ; Request main to transmit
030C 22         RET

030D D209       ErrSR:       SETB      SErrFLAG
030F 22         RET


                ;SLnRcvdR
                ;Invoked when a message received as a Slave is too long
                ;for the receive buffer.

                ;STXedR
                ;Invoked when a Slave completed transmission of its buffer.
                ;We do not expect to get here, since we do not plan to have
                ;in our system a master that will request data from this node.
                ;

                ;SRErrR
                ;Slave error event subroutine.
                ;In most applications it will not be used.
                ;

0310            SLnRcvdR:
0310            STXedR:
0310 80FB       SRErrR:      JMP       ErrSR


                ;
                ;MastNext – Master Event Routine.
                ;
                ;Invoked when a Master transaction is completed, or terminated
                ;”willingly” due to lack of acknowledge by a slave.
                ;

0312            MastNext:
0312 E524       MOV          A,MSGSTAT
0314 B42206     CJNE         A,#MTXED,MN1
0317 753750     MOV          FAILCNT,#50h
031A C208       CLR          TRQFLAG
031C 22         RET
031D            MN1:
031D 22         RET

                ;I2CDONE
                ;Called upon completion of the I2C interrupt service routine.
                ;In this example it monitors exceptions, and invokes the bus
                ;recovery routine when too many occurred.

```

```
PPCODE1         83C751 Multimaster I2C Routines                              4/14/1992          PAGE 18

031E                  I2CDONE:
031E E524             MOV         A,MSGSTAT
0320 B43208           CJNE        A,#NOTSTR,I2CD1
0323 D53705           DJNZ        FAILCNT,I2CD1
0326 753701           MOV         FAILCNT,#01h        ; Too many "illegal" i2c interrupts
0329 C2AC             CLR         EI2                 ; – shut off.
032B 22               I2CD1:      RET


                      ;*****************************************************************************
                      ;           I2C Communications Table:
                      ;*****************************************************************************



                      ; We used table driven values for clarity. one may use immediate to load
                      ; these values and save several lines of code.

                      ; Contents is used in the beginning of the main program to load
                      ; RAM location MYADDR and the I2CFG register.
                      ; The node address, in R_MYADDR, is application specific, and unique for
                      ; each device in the I2C network.
                      ; R_CTVAL depends on the crystal clock frequency.

032C 4E               R_MYADDR:   DB       4Eh        ; This node's address

032D 02               R_CTVAL:    DB       02h        ; CT1, CT0 bit values

                      ;*****************************************************************************
                      ;           Application Code Definitions
                      ;*****************************************************************************

032E 4A               PongADDR:   DB       4Ah        ; The address of the "partner" in
                                                      ; the ping–pong game.




                      END

VERSION 1.2h ASSEMBLY COMPLETE, 0 ERRORS FOUND
```

```
PPCODE1        83C751 Multimaster I2C Routines                       4/14/1992        PAGE 19

ACC . . . . . . . . . . . . . . . . . . . . . . . D ADDR   00E0H    PREDEFINED
ACKRCV8 . . . . . . . . . . . . . . . . . . C ADDR   01EDH
ADAR3  . . . . . . . . . . . . . . . . . . . . C ADDR   01A4H
ADDRRCV  . . . . . . . . . . . . . . . . . C ADDR   0050H    NOT USED
ADTXARL. . . . . . . . . . . . . . . . . . C ADDR   0186H
ADTXARL2 . . . . . . . . . . . . . . . . . C ADDR   0192H
APPFLAGS  . . . . . . . . . . . . . . . . . D ADDR   0021H
ARL  . . . . . . . . . . . . . . . . . . . . . . . B ADDR   009CH    PREDEFINED
ATN  . . . . . . . . . . . . . . . . . . . . . . . B ADDR   009EH    PREDEFINED
BCARL  . . . . . . . . . . . . . . . . . . . . .  NUMB   0010H
BCDR . . . . . . . . . . . . . . . . . . . . . . .  NUMB   0020H
BCSTP  . . . . . . . . . . . . . . . . . . . . . .  NUMB   0004H
BCSTR . . . . . . . . . . . . . . . . . . . . . .  NUMB   0008H
BCXA  . . . . . . . . . . . . . . . . . . . . . .  NUMB   0080H
BIDLE  . . . . . . . . . . . . . . . . . . . . . .  NUMB   0040H
BMRQ  . . . . . . . . . . . . . . . . . . . . . .  NUMB   0040H    NOT USED
BTIR  . . . . . . . . . . . . . . . . . . . . . . .  NUMB   0010H    NOT USED
BXSTP . . . . . . . . . . . . . . . . . . . . . .  NUMB   0001H
BXSTR . . . . . . . . . . . . . . . . . . . . . .  NUMB   0002H
CLARCV8  . . . . . . . . . . . . . . . . . . C ADDR   01F6H
CLRTI  . . . . . . . . . . . . . . . . . . . . . B ADDR   00DDH    PREDEFINED
CLSRCV8. . . . . . . . . . . . . . . . . . . C ADDR   01E2H
DESSUBAD . . . . . . . . . . . . . . . . . D ADDR   0027H
DESTADRW  . . . . . . . . . . . . . . . . D ADDR   0026H
DISMISS. . . . . . . . . . . . . . . . . . . . C ADDR   01AEH
DLY5  . . . . . . . . . . . . . . . . . . . . . . C ADDR   023DH
DRDY  . . . . . . . . . . . . . . . . . . . . . B ADDR   009DH    PREDEFINED
EA .  . . . . . . . . . . . . . . . . . . . . . . . B ADDR   00AFH    PREDEFINED
EI2.  . . . . . . . . . . . . . . . . . . . . . . . B ADDR   00ACH    PREDEFINED
ERRLED .  . . . . . . . . . . . . . . . . . . B ADDR   0091H
ERRSR . . . . . . . . . . . . . . . . . . . . . C ADDR   030DH
ETI.  . . . . . . . . . . . . . . . . . . . . . . . B ADDR   00ABH    PREDEFINED
FAILCNT.  . . . . . . . . . . . . . . . . . . D ADDR   0037H
GM2.  . . . . . . . . . . . . . . . . . . . . . . C ADDR   00CEH
GM3.  . . . . . . . . . . . . . . . . . . . . . . C ADDR   00D0H
GM4.  . . . . . . . . . . . . . . . . . . . . . . C ADDR   00E3H
GM5.  . . . . . . . . . . . . . . . . . . . . . . C ADDR   00E5H
GM6.  . . . . . . . . . . . . . . . . . . . . . . C ADDR   010BH
GM7.  . . . . . . . . . . . . . . . . . . . . . . C ADDR   0113H
GM8.  . . . . . . . . . . . . . . . . . . . . . . C ADDR   0115H
GOIDLE . . . . . . . . . . . . . . . . . . . . C ADDR   00BBH
GOMAS2 . . . . . . . . . . . . . . . . . . . C ADDR   00D8H
GOMASTER  . . . . . . . . . . . . . . . . C ADDR   00BDH
GOSLAVE.  . . . . . . . . . . . . . . . . . C ADDR   004DH
I2CD1 . . . . . . . . . . . . . . . . . . . . . . C ADDR   032BH
I2CDONE.  . . . . . . . . . . . . . . . . . . C ADDR   031EH
```

```
PPCODE1        83C751 Multimaster I2C Routines                  4/14/1992        PAGE 20

I2CFG  . . . . . . . . . . . . . . . . . . . . . D ADDR   00D8H    PREDEFINED
I2CISR . . . . . . . . . . . . . . . . . . . . . C ADDR   0023H    NOT USED
I2CON  . . . . . . . . . . . . . . . . . . . . . D ADDR   0098H    PREDEFINED
I2DAT  . . . . . . . . . . . . . . . . . . . . . D ADDR   0099H    PREDEFINED
LDELAY . . . . . . . . . . . . . . . . . . . . . C ADDR   02E6H
LDELAY1. . . . . . . . . . . . . . . . . . . . . C ADDR   02E8H
MARL . . . . . . . . . . . . . . . . . . . . . . C ADDR   017BH
MARL2  . . . . . . . . . . . . . . . . . . . . . C ADDR   0180H
MARLEND. . . . . . . . . . . . . . . . . . . . . C ADDR   0182H
MASBUF . . . . . . . . . . . . . . . . . . . . . D ADDR   002BH
MASCMD . . . . . . . . . . . . . . . . . . . . . D ADDR   0020H
MASKTABLE. . . . . . . . . . .                   C ADDR   01A6H
MASTCNT. . . . . . . . . . . . . . . . . . . . . D ADDR   0028H
MASTER . . . . . . . . . . . . . . . . . . . . . B ADDR   0099H    PREDEFINED
MASTNEXT . . . . . . . . . . . . . . . . . . . . C ADDR   0312H
MASTRQ . . . . . . . . . . . . . . . . . . . . . B ADDR   00DEH    PREDEFINED
MFAIL1 . . . . . . . . . . . . . . . . . . . . . C ADDR   02C3H    NOT USED
MGO  . . . . . . . . . . . . . . . . . . . . . . . NUMB   0020H
MMSGEND. . . . . . . . . . . . . . . . . . . . . C ADDR   0179H
MN1. . . . . . . . . . . . . . . . . . . . . . . C ADDR   031DH
MRCV . . . . . . . . . . . . . . . . . . . . . . C ADDR   0139H
MRCV2  . . . . . . . . . . . . . . . . . . . . . C ADDR   013FH
MRCVED . . . . . . . . . . . . . . . . . . . . . . NUMB   0021H
MRCVLOOP . . . . . . . . . . . . . . . . . . . . C ADDR   013DH
MSGSTAT. . . . . . . . . . . . . . . . . . . . . D ADDR   0024H
MTX. . . . . . . . . . . . . . . . . . . . . . . C ADDR   011AH
MTXED  . . . . . . . . . . . . . . . . . . . . . . NUMB   0022H
MTXLOOP. . . . . . . . . . . . . . . . . . . . . C ADDR   011BH
MTXNAK . . . . . . . . . . . . . . . . . . . . . . NUMB   0023H
MTXNOSLV . . . . . . . . . . . . . . . . . . . . . NUMB   0024H
MTXSTOP. . . . . . . . . . . . . . . . . . . . . C ADDR   0154H
MTXSTOP2 . . . . . . . . . . . . . . . . . . . . C ADDR   015CH
MTXSTOP3 . . . . . . . . . . . . . . . . . . . . C ADDR   0163H
MYADDR . . . . . . . . . . . . . . . . . . . . . D ADDR   0025H
NOACK  . . . . . . . . . . . . . . . . . . . . . C ADDR   0134H
NOGO . . . . . . . . . . . . . . . . . . . . . . C ADDR   0047H
NOSLAVE. . . . . . . . . . . . . . . . . . . . . C ADDR   012FH
NOTSTR . . . . . . . . . . . . . . . . . . . . . . NUMB   0032H
ONLED  . . . . . . . . . . . . . . . . . . . . . B ADDR   0093H
P1 . . . . . . . . . . . . . . . . . . . . . . . D ADDR   0090H    PREDEFINED
PONGADDR . . . . . . . . . . . . . . . . . . . . C ADDR   032EH
PP2. . . . . . . . . . . . . . . . . . . . . . . C ADDR   02B8H
PP22 . . . . . . . . . . . . . . . . . . . . . . C ADDR   02BEH
PP3. . . . . . . . . . . . . . . . . . . . . . . C ADDR   02CAH
PP31 . . . . . . . . . . . . . . . . . . . . . . C ADDR   02CCH
PP32 . . . . . . . . . . . . . . . . . . . . . . C ADDR   02CEH
```

```
PPCODE1          83C751 Multimaster I2C Routines                  4/14/1992         PAGE 21

PP5.  . . . . . . . . . . . . . . . . . . . . . . . C ADDR    02DCH
PPSTART.  . . . . . . . . . . . . . . . . . . C ADDR    02B5H
PPTO  . . . . . . . . . . . . . . . . . . . . . . C ADDR    02D8H     NOT USED
PSW.  . . . . . . . . . . . . . . . . . . . . . . D ADDR    00D0H     PREDEFINED
RBIT  . . . . . . . . . . . . . . . . . . . . . . C ADDR    01FFH
RBIT2  . . . . . . . . . . . . . . . . . . . . . C ADDR    0201H     NOT USED
RBIT3  . . . . . . . . . . . . . . . . . . . . . C ADDR    0208H
RBUFLEN . . . . . . . . . . . . . . . . . . .  NUMB    0004H
RC7  . . . . . . . . . . . . . . . . . . . . . . . C ADDR    024CH
RCV8   . . . . . . . . . . . . . . . . . . . . . C ADDR    01FCH
RCVERR . . . . . . . . . . . . . . . . . . . C ADDR    020EH
RCVEX  . . . . . . . . . . . . . . . . . . . . C ADDR    020DH
RDAT   . . . . . . . . . . . . . . . . . . . . . B ADDR    009FH     PREDEFINED
RECOVER.  . . . . . . . . . . . . . . . . . C ADDR    0230H
RESET . . . . . . . . . . . . . . . . . . . . . C ADDR    0278H
RESET2 . . . . . . . . . . . . . . . . . . . . C ADDR    028BH
REX.  . . . . . . . . . . . . . . . . . . . . . . C ADDR    0272H     NOT USED
RPSTRT . . . . . . . . . . . . . . . . . . . . B ADDR    0001H
R_CTVAL. . . . . . . . . . . . . . . . . . . C ADDR    032DH
R_MYADDR  . . . . . . . . . . . . . . . . C ADDR    032CH
SCL.  . . . . . . . . . . . . . . . . . . . . . . B ADDR    0080H     PREDEFINED
SDA.  . . . . . . . . . . . . . . . . . . . . . . B ADDR    0081H     PREDEFINED
SERRFLAG . . . . . . . . . . . . . . . . . B ADDR    0009H
SETMRQ . . . . . . . . . . . . . . . . . . . B ADDR    0002H
SGO  . . . . . . . . . . . . . . . . . . . . . . . .  NUMB    0010H
SLAVEN .  . . . . . . . . . . . . . . . . . . B ADDR    00DFH     PREDEFINED
SLNRCVDR  . . . . . . . . . . . . . . . . C ADDR    0310H
SLVTX . . . . . . . . . . . . . . . . . . . . . C ADDR    0084H
SMSGEND.  . . . . . . . . . . . . . . . . . C ADDR    00B3H
SMSGEND2 . . . . . . . . . . . . . . . . . C ADDR    00B9H
SP .   . . . . . . . . . . . . . . . . . . . . . . . D ADDR    0081H     PREDEFINED
SR2.  . . . . . . . . . . . . . . . . . . . . . . . C ADDR    02F9H
SR3.  . . . . . . . . . . . . . . . . . . . . . . . C ADDR    0308H
SRCV2 . . . . . . . . . . . . . . . . . . . . . C ADDR    005DH     NOT USED
SRCV3 . . . . . . . . . . . . . . . . . . . . . C ADDR    0068H
SRCVBUF. . . . . . . . . . . . . . . . . . . D ADDR    002FH
SRCVD  . . . . . . . . . . . . . . . . . . . . .  NUMB    0011H
SRCVDR .  . . . . . . . . . . . . . . . . . . C ADDR    02EFH
SRCVEND.  . . . . . . . . . . . . . . . . . C ADDR    0076H
SRCVERR.  . . . . . . . . . . . . . . . . . C ADDR    00A7H
SRCVSTO. . . . . . . . . . . . . . . . . . . C ADDR    0066H
SRERR . . . . . . . . . . . . . . . . . . . . . .  NUMB    0014H
SRERRR .  . . . . . . . . . . . . . . . . . . C ADDR    0310H
SRLNG  . . . . . . . . . . . . . . . . . . . . .  NUMB    0012H
STACKSAVE . . . . . . . . . . . . . . . . D ADDR    002AH
STP.  . . . . . . . . . . . . . . . . . . . . . . . B ADDR    009AH     PREDEFINED
```

```
PPCODE1          83C751 Multimaster I2C Routines                    4/14/1992        PAGE 22

STR.   . . . . . . . . . . . . . . . . . . . . . . B ADDR   009BH    PREDEFINED
STSTRW .  . . . . . . . . . . . . . . . . . . . . C ADDR   0055H
STX2  . . . . . . . . . . . . . . . . . . . . . . C ADDR   0085H    NOT USED
STXBUF .  . . . . . . . . . . . . . . . . . . . . D ADDR   0033H
STXED . . . . . . . . . . . . . . . . . . . . . .  NUMB    0013H
STXEDR .  . . . . . . . . . . . . . . . . . . . . C ADDR   0310H
STXERR .  . . . . . . . . . . . . . . . . . . . . C ADDR   00AEH    NOT USED
STXLP  . . . . . . . . . . . . . . . . . . . . . . C ADDR   0093H
SUBADD . . . . . . . . . . . . . . . . . . . . . . B ADDR   0000H
TI1.   . . . . . . . . . . . . . . . . . . . . . . C ADDR   0219H    NOT USED
TI2.   . . . . . . . . . . . . . . . . . . . . . . C ADDR   021CH    NOT USED
TI3.   . . . . . . . . . . . . . . . . . . . . . . C ADDR   0223H
TI4.   . . . . . . . . . . . . . . . . . . . . . . C ADDR   0225H
TIISR  . . . . . . . . . . . . . . . . . . . . . . C ADDR   0211H
TIMERI .  . . . . . . . . . . . . . . . . . . . . C ADDR   001BH    NOT USED
TIRUN  . . . . . . . . . . . . . . . . . . . . . . B ADDR   00DCH    PREDEFINED
TITOCNT.  . . . . . . . . . . . . . . . . . . . . D ADDR   0029H
TOGCNT .  . . . . . . . . . . . . . . . . . . . . D ADDR   0038H
TOGLED .  . . . . . . . . . . . . . . . . . . . . B ADDR   0090H
TRQFLAG.  . . . . . . . . . . . . . . . . . . . . B ADDR   0008H
XMADDR .  . . . . . . . . . . . . . . . . . . . . C ADDR   01C5H
XMBEX  . . . . . . . . . . . . . . . . . . . . . . C ADDR   01E1H
XMBIT . . . . . . . . . . . . . . . . . . . . . . C ADDR   01D0H
XMBIT2 . . . . . . . . . . . . . . . . . . . . . . C ADDR   01D2H
XMBYTE . . . . . . . . . . . . . . . . . . . . . . C ADDR   01CEH
XRETI  . . . . . . . . . . . . . . . . . . . . . . C ADDR   004CH
```