APPLICATION NOTE

EIE/AN91007

I^2C driver routines for 8XC751/2 microcontrollers

Author: J.C.P.M. Pijnenburg, Eindhoven

1991 Jul

Philips
Semiconductors

PHILIPS

I²C driver routines for 8XC751/2 microcontrollers EIE/AN91007

Author: J.C.P.M. Pijnenburg, Eindhoven

1. INTRODUCTION

This report describes the I²C drivers which are written for the 8xC751/2. The report describes not only how to use the routines, but also the structure of the software. The software is written around a set of basic routines and a message handler. The message handler does not contain any specific 8xC751 code, so the software can be easily rewritten for any other bit level I²C interface by rewriting the set of basic routines. In the rest of this report, when 8xC751 is written, it means 8xC751/2.

The package supports also the multimaster features of the I²C bus.

The maximum bit rate possible when using those routines is approximately 70Kbit/sec.

References:
- The I²C-bus specification: 9398 358 10011
- 80C51-based 8-bit microcontrollers Data Handbook IC20
- PLM51 I²C Software interface I2C51: ETV/AN89004

2. GENERAL

2.1 Memory Usage & File Structure

The driver software consists of 3 main parts:
- I²C message handler
- I²C basic routines
- I²C slave routines

During I²C usage it claims register bank 1, however, register bank 1 does not contain any static I²C data and can be used by the application program outside the I²C routines (this data will be destroyed by I²C routines). The accumulator is also modified during I²C transfer.

The message handler uses a Message Control Block which consists of 8 bytes RAM. In those bytes, the following parameters are stored:

For block 1: I2C_ADDR_1, BUF_LEN and BUF_PTR_1

For block 2: I2C_ADDR_2, BUF_LEN and BUF_PTR_2

2 bytes of bit addressable RAM for STATUS and CONTROL information

The STATUS byte is returned into the accumulator. If you do not need a detailed status, you can test the carry bit, this is a copy of the I2C_ERROR bit of the status register (returned in the acc.). The status register contains the following information:

BIT	NAME	FUNCTION
0	RETRY_0	|
1	RETRY_1	|– Retry counter (0..7), as given during I2C_INIT
2	RETRY_2	|
3	I2C_ERR	I2C error if set (also available in carry)
4	TIME_ERR	Bus timeout occurred if set
5	RECOVER	– (no value for user) always 0
6	BUS_RECOVERED	If set, bus K recovered after timeout
7	NO_ACK	No acknowledge received

The slave function uses 2 bytes of RAM, those contain the own slave address (OWN_SLV_ADDR) and a pointer the slave transmit/receive buffer of the 8xC751. This is the buffer from/in which the 8xC751 gets/stores the data bytes in slave mode.

The I²C module is built around a message handler which calls basic functions such as I2C_TRX_BYTE and I2C_START. Each function calls the message handler after loading the correct mask into the I2C_CTRL byte.

FILENAME	FUNCTION	INCLUDE/LINK	CODE SIZE (BYTE)
I2C_DATA.GLO	I²C global data definitions	I, each I²C function and assembler main	0
I2C_DATA.LOC	I²C local data definitions	I, each I²C function	0
I2C_CODE.GLO	I²C global function definitions	I, assembler main	0
I2C_INIT.ASM	Init_I2C (does not use message handler)	Link	22
I2C_DEF.ASM	Define MCB & _I2C_xxx_BYTEs	Link	0
I2C_HAND.ASM	I²C Message handler	Link	144
I2C_BASI.ASM	I²C basic functions, and TI interrupt handling	Link	293
I2C_TDEV.ASM	I²C Test_Device	Link, if used	5
I2C_WRIT.ASM	I²C Write	Link, if used	5
I2C_WSUB.ASM	I²C Write_Sub	Link, if used	5
I2C_WSWI.ASM	I²C Write_Sub_SWinc & Write_Mem	Link, if used	36
I2C_WSUW.ASM	I²C Write_Sub_Write	Link, if used	5
I2C_WSUR.ASM	I²C Write_Sub_Read	Link, if used	5
I2C_WCOW.ASM	I²C Write_Com_Write	Link, if used	11
I2C_WREW.ASM	I²C Write_Rep_Write	Link, if used	5
I2C_WRER.ASM	I²C Write_Rep_Read	Link, if used	5
I2C_READ.ASM	I²C Read and Read_Status	Link, if used	11
I2C_RSUB.ASM	I²C Read_Sub	Link, if used	5
I2C_RRER.ASM	I²C Read_Rep_Read	Link, if used	5
I2C_RREW.ASM	I²C Read_Rep_Write	Link, if used	5
I2C_SLAV.ASM	I²C Slave Function	Link	56

The total memory usage for the full package is

ROM		:	520 (single function) to 623 (all functions) bytes
RAM	byte addressable	:	8 bytes
	bit addressable	:	2 bytes
	register bank 1	:	8 bytes

The message handler, causes the other functions to be very small, to further reduce the code, all functions are placed in separate modules, which are put into a library I2C_751.LIB. If this library is linked to an application program, only the object modules which are used by the application program are linked in the output file.

The I2C_CODE.H file contains the references to the separate functions (EXTRN CODE definitions). The use must not include this file into main, but only copy the definitions which he needs into the source file. If this file is included, all functions will be linked, the library approach is of no use in this case.

2.2 Retries

During initialization, the user defines whether he wants to use retries or not. If an I²C message fails, and retries >=0, the program restarts the message. This is done for at most 7 times. If the message remains unsuccessful, the message handler returns to the main program, indicating that the message has failed (carry set).

2.3 Error Handling

In case of an error while operating as master, the program returns to the message handler. The message handler decides whether to invoke a retry or to return to the main program.

The I²C interface of the 8xC751 generates a timeout interrupt if the bus hangs for more than 1022 cycles, in this case, if the 8xC751 is master (RECOVER = 1), a bus recover routine is started, if the 8xC751 is not master, the I²C bus is released. Retries are only invoked in the master situation.

2.4 Development Tools

The following software tools from Tasking/BSO are used for program development:

– OM4142 Cross Assembler 8051 for DOS: V3.0b
– OM4144 PL/M 8051 Compiler for DOS: V3.0a
– OM4136 C8051 Compiler for DOS: V1.1a
– OM4129 XRAY51 debugger: V1.4c

3. MASTER ROUTINES

3.1 Message Handler

To make the I²C protocols as described in paragraphs 3.2 to 3.15, and I²C message handler is written. The message control block (I2C_MCB) together with the I²C control byte (I2C_CTRL) form the input for the message handler. The I2C_MCB includes 6 bytes of data, containing:

- I2C_ADDR1, first address in the protocol
- BUF_LEN_1, length of the first data buffer
- BUF_PTR_1, pointer to the first data buffer
- I2C_ADDR2, second address in the protocol
- BUF_LEN_2, length of the second data buffer
- BUF_PTR_2, pointer to the second data buffer

The I2C_CTRL byte is bit addressable. It contains 8 bits that determine the flow through the message handler. This byte must be loaded with the corresponding mask before starting the message handler.

The I2C_CTRL byte contains the following bits:

- REP_STRT_BLK1 must we send a repeated start before the first data block? (0=NO, 1=YES)
- RWN_BLK1 read (1) or write (0) the first block of data
- ADDR2 is there a second address in the protocol? (0=NO, 1=YES)
- ADDR2_SUB is the 2nd address a sub address, only relevant if ADDR2=1.
- BLOCK2 is there a second block of data in the protocol? (0=NO, 1=YES)
- RWN_BLK1 read (1) or write (0) the first block of data
- REP_STRT_BLK1 must we send a repeated start before the second data block?
- TEST_DEVICE is it the test device protocol

When an I²C protocol is handled successfully by the message handler, it returns control to the main program; if not it can do a retry by resending the message (maximum 5 retries are possible).

The message handler return value is stored in the I2C_STAT byte. The I2C_ERROR bit indicates whether the transfer has succeeded.

NOTE: It is better to copy this byte into Acc before returning the control to the main program, this way a byte can be saved (I2C_STAT can be placed in register bank 1) and the main program can do a JZ/JNZ test (must be changed).

3.2 I2C_INIT

Description

Init_I2C must be called after RESET, before any procedure is called. The I²C interface and I²C interrupt will be enabled (STEB ETI,EI2 and EA). Own_Slave_Address is passed to Init_I2C for use as slave. Slave_Sub_Address is the pointer to a DATA buffer that is used for data transfer in slave mode. When used as master in a single master system, these parameters are not used. Retry is the number of retries on messages when an error occurs. 0 means no retry (just 1 attempt to send a message), while the maximum amount of retries is 7.

I²C Protocol

none (no action at I²C bus)

Calling Sequence

C	: I2C_INIT(Own_Slv_Addr,Slv_Buf_Addr,Retry);
PL/M51	: I2C_INIT(Own_Slv_Addr,Slv_Buf_Addr,Retry);
Assembler	: %I2C_INIT(Own_Slv_Addr,Slv_Buf_Addr,Retry); (macro call)

Parameters

Own_Slave_Adr	: 8xC751 own slave address
Slave_Buffer_Adr	: Base address of buffer, to transmit data from, or receive data in, when 8xC751 is in slave mode.
Retry	: Number of times to do a retry in case of an error. 0 = No Retry, maximum retries is 7.

NOTE:

The Init_I2C function enables the I²C watchdog timer interrupt (TI). This watchdog generates an interrupt when during and I²C transfer, SCL is held longer than 1022 machine cycles (ca. 760µs @ 16MHz). If this time is too short for your application, you can disable the TI (CLR ETI). In this case, the main program must check if a bus hangup occurs, and take proper action when the bus is hangup.

3.3 I2C_TEST_DEVICE

Description
I2C_Test_Device just sends the slave address to the I²C bus. It can be used to check the presence of a device on the I²C bus.

I²C Protocol
Slv_W : Slave_Adr + Write bit

S	Slv_W	A	P

Device is present

S	Slv_W	N	P

Device is not present

Calling Sequence
C : I2C_TEST_DEVICE(Slv_Addr);
PL/M51 : I2C_TEST_DEVICE(Slv_Addr);
Assembler : %I2C_TEST_DEVICE(Slv_Addr);
(macro call)

Parameters
Slave_Adr : Slave address of the device to be tested.

3.4 I2C_WRITE

Description
I2C_Write is the most basic procedure to write a message to a slave device.

I²C Protocol
Slv_W : Slave_Adr + Write bit
D0..Dn : Data bytes

S	Slv_W	A	D0	A	D1	A	D2	A	. . .	A	Dn	A	P

Calling Sequence
C : I2C_WRITE(Slv_Addr,Count,Source_Ptr);
PL/M51 : I2C_WRITE(Slv_Addr,Count,Source_Ptr);
Assembler : %I2C_WRITE(Slv_Addr,Count,Source_Ptr);

Parameters
Slave_Adr : Slave address of the device to write to.
Count : Number of bytes to transmit (D0 .. Dn, n = count – 1)
Source_Ptr : Pointer to data buffer, to transmit bytes from.
(macro call)

3.5 I2C_WRITE_SUB

Description
I2C_Write_Sub writes a message preceded by a sub-address to a slave device.

I²C Protocol
Slv_W : Slave_Adr + Write bit
Sub : Sub_Adr
D0..Dn : Data bytes

S	Slv_W	A	Sub	A	D0	A	D1	A	. . .	A	Dn	A	P

Calling Sequence
C : I2C_WRITE_SUB(Slv_Addr,Count,Source_Ptr,Sub_Addr);
PL/M51 : I2C_WRITE_SUB(Slv_Addr,Count,Source_Ptr,Sub_Addr);
Assembler : %I2C_WRITE_SUB(Slv_Addr,Count,Source_Ptr,Sub_Addr);
(macro call)

Parameters
Slave_Adr : Slave address of the device to write to.
Count : Number of bytes to transmit (D0 .. Dn, n = count – 1)
Source_Ptr : Pointer to data buffer, to transmit bytes from.
Sub_Adr : Sub address.

3.6 I2C_WRITE_SUB_SWINC

Description
Some I²C devices addressed with a sub-address do not automatically increment the sub-address after reception of each byte.
I2C_Write_Sub_SWInc can be used for such devices the same way as I2C_Write_Sub is used. I2C_Write_Sub_SWInc splits up the message in smaller messages and increments the sub-address itself.

I²C Protocol
Slv_W : Slave_Adr + Write bit
Sub+x : Sub_Adr+x
D0..Dn : Data bytes

S	Slv_W	A	Sub+0	A	D0	P

S	Slv_W	A	Sub+1	A	D1	P

. . . .

S	Slv_W	A	Sub+n	A	Dn	P

Calling Sequence
C : I2C_WRITE_SUB_SWINC(Slv_Addr,Count,Source_Ptr,Sub_Addr);
PL/M51 : I2C_WRITE_SUB_SWINC(Slv_Addr,Count,Source_Ptr,Sub_Addr);
Assembler : %I2C_WRITE_SUB_SWINC(Slv_Addr,Count,Source_Ptr,Sub_Addr);
(macro call)

Parameters
Slave_Adr : Slave address of the device to write to.
Count : Number of bytes to transmit (D0 .. Dn, n = count – 1)
Source_Ptr : Pointer to data buffer, to transmit bytes from.
Sub_Adr : Sub address.

3.7 I2C_WRITE_MEMORY

Description

I²C Non-Volatile Memory devices (such as PCF8582) need an additional delay after writing a byte to it. I2C_Write_Memory can be used to write to such devices the same way I2C_Write_Sub is used. I2C_Write_Memory splits up the message in smaller messages and increments the sub-address itself. After transmission of each message, a delay of 40 milliseconds (f_{XTAL} = 16MHz) is inserted.

I²C Protocol

Slv_W : Slave_Adr + Write bit
Sub+x : Sub_Adr+x
D0..Dn : Data bytes

S	Slv_W	A	Sub+0	A	D0	P

40 mS

S	Slv_W	A	Sub+1	A	D1	P

40 mS

. . . .

40 mS

S	Slv_W	A	Sub+n	A	Dn	P

40 mS

Calling Sequence

C : I2C_WRITE_MEMORY(Slv_Addr,Count,Source_Ptr,Sub_Addr);
PL/M51 : I2C_WRITE_MEMORY(Slv_Addr,Count,Source_Ptr,Sub_Addr);
Assembler : %I2C_WRITE_MEMORY(Slv_Addr,Count,Source_Ptr,Sub_Addr);
(macro call)

Parameters

Slave_Adr : Slave address of the device to write to.
Count : Number of bytes to transmit (D0 .. Dn, n = count – 1)
Source_Ptr : Pointer to data buffer, to transmit bytes from.
Sub_Adr : Sub address.

3.8 I2C_WRITE_SUB_WRITE

Description
I2C_Write_Sub_Write writes 2 data blocks preceded by a sub-address in one message to a slave device. This procedure can be used for devices that need an extended addressing method, without the need to put all data into one large buffer. Such a device is the ECCT (I²C controlled teletext device; see example).

I²C Protocol
Slv_W : Slave_Adr + Write bit
Sub : Sub_Adr
D1.0..D1.n : Data bytes in first block
D2.0..D2.p : Data bytes in second block

S	Slv_W	A	Sub	A	D1.0	A	D1.1	A	. . .	A	D1.n	A	D2.0	A	. . .	A	D2.p	A	P

Calling Sequence
C : I2C_WRITE_SUB_WRITE(Slv_Addr,Count_1,Source_Ptr_1,Sub_Addr,Count_2,Source_Ptr_2);
PL/M51 : I2C_WRITE_SUB_WRITE(Slv_Addr,Count_1,Source_Ptr_1,Sub_Addr,Count_2,Source_Ptr_2);
Assembler : %I2C_WRITE_SUB_WRITE(Slv_Addr,Count_1,Source_Ptr_1,Sub_Addr,Count_2,Source_Ptr_2);
(macro call)

Parameters
Slave_Adr_1 : Slave address of the device to write to.
Count_1 : Number of bytes to transmit in first block (D1.0 .. D1.n, n = count_1 – 1)
Source_Ptr_1 : Pointer to first block of data to transmit.
Sub_Adr : Sub address.
Count_2 : Number of bytes to transmit in second block (D2.0 .. D2.p, p = count_2 – 1)
Source_Ptr_2 : Pointer to second block of data to transmit.

3.9 I2C_WRITE_SUB_READ

Description
I2C_Write_Sub_Read writes a data block preceded by a sub-address, generates an I²C restart condition, and reads a data block. This procedure can be used for devices that need an extended addressing method. Such a device is the ECCT.

I²C Protocol
Slv_W : Slave_Adr + Write bit
Slv_R : Slave_Adr + Read bit
Sub : Sub_Adr
D1.0..D1.n : Data bytes in first block (write)
D2.0..D2.p : Data bytes in second block (read)

S	Slv_W	A	Sub	A	D1.0	A	D1.1	A	. . .	A	D1.n	A	S	Slv_R	A	D2.0	A	D2.1	A	. . .	A	D2.p	N	P

Calling Sequence
C : I2C_WRITE_SUB_READ(Slv_Addr,Count_1,Source_Ptr,Sub_Addr,Count,Dest_Ptr);
PL/M51 : I2C_WRITE_SUB_READ(Slv_Addr,Count_1,Source_Ptr,Sub_Addr,Count,Dest_Ptr);
Assembler : %I2C_WRITE_SUB_READ(Slv_Addr,Count_1,Source_Ptr,Sub_Addr,Count,Dest_Ptr);
(macro call)

Parameters
Slave_Adr_1 : Slave address of the device to write and read to/from.
Count_1 : Number of bytes to transmit (D1.0 .. D1.n, n = count – 1)
Source_Ptr_1 : Pointer to first block of data to transmit.
Sub_Adr : Sub address.
Count_2 : Number of bytes to transmit in second block (D2.0 .. D2.p, p = count_2 – 1)
Dest_Ptr_2 : Pointer buffer to receive second block of data in.

3.10 I2C_WRITE_COM_WRITE

Description

I2C_Write_Com_Write writes two data blocks from different data buffers in one message to a slave receiver. This procedure can be used for devices where the message consists of 2 different data blocks. Such devices are, for instance, LCD-drivers, where the first part of the message consists of addressing and control information, and the second part is the data string to be displayed.

I²C Protocol

Slv_W : Slave_Adr + Write bit
D1.0..D1.n : Data bytes in first block (write)
D2.0..D2.p : Data bytes in second block (write)

S	Slv_W	A	D1.0	A	D1.1	A	D1.2	A	. . .	A	D1.n	A	D2.0	A	. . .	A	D2.p	A	P

Calling Sequence

C : I2C_WRITE_COM_WRITE(Slv_Addr,Count_1,Source_Ptr_1,Count_2,Source_Ptr_2);
PL/M51 : I2C_WRITE_COM_WRITE(Slv_Addr,Count_1,Source_Ptr_1,Count_2,Source_Ptr_2);
Assembler : %I2C_WRITE_COM_WRITE(Slv_Addr,Count_1,Source_Ptr_1,Count_2,Source_Ptr_2);
(macro call)

Parameters

Slave_Adr : Slave address of the device to write to.
Count_1 : Number of bytes to transmit in first block (D1.0 .. D1.n, n = count_1 – 1)
Source_Ptr_1 : Pointer to first block of data to transmit.
Count_2 : Number of bytes to transmit in second block (D2.0 .. D2.p, p = count_2 – 1)
Source_Ptr_2 : Pointer to second block of data to transmit.

3.11 I2C_WRITE_REP_WRITE

Description

Two data strings are sent to separate slave devices, separated with a repeat START condition. This has the advantage that the bus does not have to be released with a STOP condition before the transfer from the second slave.

I²C Protocol

Slv1W : Slave_Adr _1 + Write bit
Slv2W : Slave_Adr_2 + Write bit
D1.0..D1.n : Data bytes in first block (write to first slave)
D2.0..D2.p : Data bytes in second block (write to second slave)

S	Slv1W	A	D1.0	A	D1.1	A	D1.2	A	. . .	A	D1.n	A	P

Calling Sequence

C : I2C_WRITE_REP_WRITE(Slv_Addr,Count_1,Source_Ptr_1,Sub_Addr,Count_2,Source_Ptr_2);
PL/M51 : I2C_WRITE_REP_WRITE(Slv_Addr,Count_1,Source_Ptr_1,Sub_Addr,Count_2,Source_Ptr_2);
Assembler : %I2C_WRITE_REP_WRITE(Slv_Addr,Count_1,Source_Ptr_1,Sub_Addr,Count_2,Source_Ptr_2);
(macro call)

Parameters

Slave_Adr_1 : Slave address of first device to write to.
Count_1 : Number of bytes to transmit in first block (D1.0 .. D1.n, n = count_1 – 1)
Source_Ptr_1 : Pointer to first block of data to transmit.
Slave_Adr_2 : Slave address of second device to write to.
Count_2 : Number of bytes to transmit in second block (D2.0 .. D2.p, p = count_2 – 1)
Source_Ptr_2 : Pointer to second block of data to transmit.

3.12 I2C_WRITE_REP_READ

Description
A data string is sent and received to/from two separate slave devices, separated with a repeat START condition. This has the advantage that the bus does not have to be released with a STOP condition before the transfer from the second slave.

I²C Protocol
Slv1W : Slave_Adr _1 + Write bit
Slv2R : Slave_Adr_2 + Read bit
D1.0..D1.n : Data bytes in first block (write to first slave)
D2.0..D2.p : Data bytes in second block (write to second slave)

S	Slv1W	A	D1.0	A	D1.1	A	D1.2	A	. . .	A	D1.n	A	S	Slv2R	A	D2.0	A	D2.1	A	. . .	A	D2.p	N	P

Calling Sequence
C : I2C_WRITE_REP_READ(Slv_Addr,Count_1,Source_Ptr,Sub_Addr,Count_2,Dest_Ptr);
PL/M51 : I2C_WRITE_REP_READ(Slv_Addr,Count_1,Source_Ptr,Sub_Addr,Count_2,Dest_Ptr);
Assembler : %I2C_WRITE_REP_READ(Slv_Addr,Count_1,Source_Ptr,Sub_Addr,Count_2,Dest_Ptr);
(macro call)

Parameters
Slave_Adr_1 : Slave address of first device to write to.
Count_1 : Number of bytes to transmit in first block (D1.0 .. D1.n, n = count_1 – 1)
Source_Ptr_1 : Pointer to first block of data to transmit.
Slave_Adr_2 : Slave address of second device to read from.
Count_2 : Number of bytes to transmit in second block (D2.0 .. D2.p, p = count_2 – 1)
Dest_Ptr_2 : Pointer buffer to receive second block of data in.

3.13 I2C_READ

Description
I2C_Read is the most basic procedure to read a message from a slave device.

I²C Protocol
Slv_R : Slave_Adr + Read bit
D0 .. Dn : Data bytes

S	Slv_R	A	D0	A	D1	A	D2	A	. . .	A	Dn	A	P

Calling Sequence
C : I2C_READ(Slv_Addr,Count,Dest_Ptr);
PL/M51 : I2C_READ(Slv_Addr,Count,Dest_Ptr);
Assembler : %I2C_READ(Slv_Addr,Count,Dest_Ptr);
(macro call)

Parameters
Slave_Adr : Slave address of the device to be tested.
Count : Number of bytes to transmit (D0 .. Dn, n = count – 1)
Dest_Ptr : Pointer to data buffer, to receive bytes in.

3.14 I2C_READ_STATUS

Description
Several I²C devices can send a one byte status-word via the bus. I2C_Read_Status can be used for this purpose. I2C_Read_Status works the same way as I2C_Read, but the user does not have to pass a count parameter.

I²C Protocol
Slv_R : Slave_Adr + Read bit
Status : Status bytes

S	Slv_R	A	Status	A	P

Calling Sequence
C : I2C_READ_STATUS(Slv_Addr,Dest_Ptr);
PL/M51 : I2C_READ_STATUS(Slv_Addr,Dest_Ptr);
Assembler : %I2C_READ_STATUS(Slv_Addr,Dest_Ptr);
(macro call)

Parameters
Slave_Adr : Slave address of the device to be tested.
Count : Number of bytes to transmit (D0 .. Dn, n = count – 1)
Dest_Ptr : Pointer to data buffer, to receive status byte in.

3.15 I2C_READ_SUB

Description
I2C_Read_Sub reads a message from a slave device, preceded by a write of the sub-address. Between writing the sub-address and reading the message, an I²C restart condition is generated without releasing the bus. This prevents other masters from accessing the slave device in between and overwriting the sub-address.

I²C Protocol
Slv_W : Slave_Adr + Write bit
Slv_R : Slave_Adr + Read bit
Sub : Sub_Adr

S	Slv_W	A	Sub	A	S	Slv_R	A	D1	A	. . .	A	Dn	N	P

Calling Sequence
C : I2C_READ_SUB(Slv_Addr,Count,Dest_Ptr,Sub_Addr);
PL/M51 : I2C_READ_SUB(Slv_Addr,Count,Dest_Ptr,Sub_Addr);
Assembler : %I2C_READ_SUB(Slv_Addr,Count,Dest_Ptr,Sub_Addr);
(macro call)

Parameters
Slave_Adr : Slave address of the device to be tested.
Count : Number of bytes to transmit (D0 .. Dn, n = count – 1)
Dest_Ptr : Pointer to data buffer, to receive bytes in.
Sub_Adr : Sub address.

3.16 I2C_READ_REP_READ

Description
Two data strings are read from separate slave devices, separated with a repeat START condition. This has the advantage that the bus does not have to be released with a STOP condition before the transfer from the second slave.

I²C Protocol
Slv1R : Slave_Adr_1 + Read bit
Slv2R : Slave_Adr_2 + Read bit
D1.0 .. D1.n : Data bytes in first block (read from first slave)
D2.0 .. D2.p : Data bytes in second block (read from second slave)

S	Slv1R	A	D1.0	A	D1.1	A	D1.2	A	. . .	A	D1.n	N	S	Slv2R	A	D2.0	A	D2.1	A	. . .	A	D2.p	N	P

Calling Sequence
C : I2C_READ_REP_READ(Slv_Addr,Count_1,Dest_Ptr_1,Sub_Addr,Count_2,Dest_Ptr_2);
PL/M51 : I2C_READ_REP_READ(Slv_Addr,Count_1,Dest_Ptr_1,Sub_Addr,Count_2,Dest_Ptr_2);
Assembler : %I2C_READ_REP_READ(Slv_Addr,Count_1,Dest_Ptr_1,Sub_Addr,Count_2,Dest_Ptr_2);
(macro call)

Parameters
Slave_Adr_1 : Slave address of first device to write to.
Count_1 : Number of bytes to transmit in first block (D1.0 .. D1.n, n = count_1 – 1)
Dest_Ptr_1 : Pointer buffer to receive first block of data in.
Sub_Adr_2 : Slave address of second device to read from.
Count_2 : Number of bytes to transmit in second block (D2.0 .. D2.p, p = count_2 – 1)
Dest_Ptr_2 : Pointer buffer to receive second block of data in.

3.17 I2C_READ_REP_WRITE

Description
A data string is received and sent from/to two separate slave devices, separated with a repeat START condition. This has the advantage that the bus does not have to be released with a STOP condition before the transfer from the second slave.

I²C Protocol
Slv1R : Slave_Adr_1 + Read bit
Slv2W : Slave_Adr_2 + Write bit
D1.0 .. D1.n : Data bytes in first block (read from first slave)
D2.0 .. D1.p : Data bytes in second block (read from second slave)

S	Slv1R	A	D1.0	A	D1.1	A	D1.2	A	. . .	A	D1.n	N	S	Slv2W	A	D2.0	A	D2.1	A	. . .	A	D2.p	A	P

Calling Sequence
C : I2C_READ_REP_WRITE(Slv_Addr,Count_1,Dest_Ptr_1,Sub_Addr,Count_2,Source_Ptr);
PL/M51 : I2C_READ_REP_WRITE(Slv_Addr,Count_1,Dest_Ptr_1,Sub_Addr,Count_2,Source_Ptr);
Assembler : %I2C_READ_REP_WRITE(Slv_Addr,Count_1,Dest_Ptr_1,Sub_Addr,Count_2,Source_Ptr);
(macro call)

Parameters
Slave_Adr_1 : Slave address of first device to write to.
Count_1 : Number of bytes to transmit in first block (D1.0 .. D1.n, n = count_1 – 1)
Dest_Ptr_1 : Pointer buffer to receive first block of data in.
Sub_Adr_2 : Slave address of second device to read from.
Count_2 : Number of bytes to transmit in second block (D2.0 .. D2.p, p = count_2 – 1)
Dest_Ptr_2 : Pointer buffer to transmit second block of data from.

4. SLAVE ROUTINES

The slave-mode protocol is very application dependent. In this note the basic slave-receive and slave-transmit routines are given and should be considered as examples. The user may, for instance, send NO_ACK after receiving a number of bytes to signal to the master-transmitter that a data buffer is full. A listing of the slave routines is given in Appendix III.

The I²C slave function has two entries:

1. The I²c interrupt.
   This can only occur at an idle slave, because when a transmission is in progress, the I²C interrupt is disabled.
2. Through the master routines.
   During transmission of a slave-address in master-mode, arbitration is lost to another master. The interface must then switch to slave-receiver mode to check if this other master wants to address the 8xC751 I²C interface. If the 8xC751 recognizes his own slave address, the slave mode routines are entered at labels I2C_SLV_TRX or I2C_SLV_RCV.

Interfacing the master routines, if the user wants to adapt the slave routines to his own needs, he has to keep in mind that the master routines use the I2C_SLV_TRX and I2C_SLV_RCV entries. The I²C slave routines are entered after the acknowledge has been sent, therefore the ATN flag will be set when entering the slave routines at I2C_SLV_TRX or I2C_SLV_RVC.

The slave routines, as given, make use of a single data buffer. When addressed as slave transmitter, data bytes from the data buffer are transmitted over the I²C bus until a not acknowledge or stop is received. When addressed as slave receiver, the data from the I²C bus is received into the data buffer until a not acknowledge or a stop is received.

The data buffer is initialized during the Init_I2C function, one of the parameters of this function is the pointer to the data buffer (SLV_BUF_PTR DS 1).

4.1 Slave Transmitter

The slave transmitter function transmits data bytes from the 8xC751 data buffer (ACALL I2C_TRX_BYTE) until a not acknowledge or a stop is received. The function is also exit on an I²C error. The function is exit with the ATN bit set.

4.2 Slave Receiver

The slave receiver function receivers data bytes into the 8xC751 data buffer (ACALL I2C_RCV_BYTE) until a stop is received. The function is also exit on an I²C error. The function is exit with the ATN bit set. If a byte has been received, an acknowledge is sent.

5. EXAMPLES

5.1 Introduction

Some examples are given on how to use the I²C routines in an application program. Examples are given for an assembly, PL/M and C programs. The program displays time form the PCF8583P clock/calendar/RAM on an LCD display driven by the PCF8577. The example can be executed on the OM4151 I²C evaluation board.

5.2 Using the Routines in Assembly Sources

Appendix VII shows the listing of the example program. The most important aspect when using the I²C routines is preparing the input parameters before the sub-routine call. The parameters must be transferred to the MCB (Message Control Block). Below are 2 examples of how to transfer the necessary parameters to MCB (_I2C_Read and _I2C_Write_Sub_Read).

```
MOV _I2C_MCB,#Slave_Adr
MOV _I2C_MCB+1,#Count_1
MOV _I2C_MCB+2,#Dest_Ptr_1
ACALL _I2C_READ
```

```
MOV _I2C_MCB,#S1_Adr
MOV _I2C_MCB+1,#Cnt_1
MOV _I2C_MCB+2,#S_Ptr_1
MOV _I2C_MCB+3,#Sub_Adr
MOV _I2C_MCB+4,#Cnt_2
MOV _I2C_MCB+5,#S_Ptr_2
ACALL _I2C_WRITE_SUB_READ
```

Note that the order of defining the parameters is the same as in PL/M– and C-calls (Calling sequences in paragraphs 3.2 to 3.17). An easier way to call the routines is to make a macro that includes the to transfer of the parameters.

The example program makes use of macros. I2C_Read is then called in the following way:

```
%I2C_READ(Slave_Adr,Count_1,Source_Ptr_1);
```

Note that in the listing the macro call is replaced by the contents of the macro.

The macro must be written as follows:

```
%* DEFINE (I2C_READ(Slave_Adr,Count_1,Dest_Ptr_1))
(
  MOV _I2C_MCB,#%Slave_Adr
  MOV _I2C_MCB+1,#%Count_1
  MOV _I2C_MCB+2,#%Dest_Ptr_1
  ACALL _I2C_READ
)
```

File I2C_MAC.DEF contains the macro calls for the routines as described in paragraphs 3.2 to 3.17. This file should be included in all assembler modules in which calls to the I²C routines are made.

The file I2C_CODE.GLO contains the global function definitions (EXTRN CODE) of the I²C functions, copy the ones you need into your application. The file I2C_DATA.GLO contains the global data definitions of the I²C functions. Therefore, this file must also be included in all assembler modules in which calls to the I²C routines are made.

All I²C routines return a status into the CY-bit. If the CY-bit is set, and error has occurred.

5.3 Using the Routines in PL/M-51 Sources

Appendix VIII shows the listing of the example program in PL/M-51. All procedures return a BIT value. The file I2C_PL/M.H contains the procedure declarations, this file can be included in the modules which call I²C routines. The routines are used the same way as in the examples of paragraph 5.2.

5.4 Using the Routines in C Sources

Appendix IX shows the listing of the example program in C. All functions are return a bit value. The file I2C_C.H contains the function prototypes, this file can be included in the modules which call I²C routines. The routines are used the same way as in the examples of paragraph 5.2.

Read.Me

```
*=============================================================================*
*                                                                             *
*    PACKAGE: I2C drivers for 8xC751/2 microcontroller                        *
*                                                                             *
*    DESCRIPTION: To use the package just link the library: I2c_751.lib       *
*                 to your application program                                 *
*                                                                             *
*    NOTES: If you use the package with assembler sources, you must include   *
*           \USER\INCLUDE\I2C_DATA.GLO and \USER\INCLUDE\I2C_MAC.DEF into     *
*           your main application(s).                                         *
*           \USER\INCLUDE\I2C_CODE.GLO contains external code definitions,    *
*           select the ones you need and copy them into your main application*
*           If you include this file, the linker assumes that you use all I2C*
*           functions and therefore links the complete package to your        *
*           application (in this case the library approach is of no use!)     *
*                                                                             *
*           If you use the package with PLM sources, you must include         *
*           \USER\INCLUDE\I2C_PLM.H in each file which uses an I2C function   *
*                                                                             *
*           If you use the package with C sources, you must include           *
*           \USER\INCLUDE\I2C_PLM.C in each file which uses an I2C function   *
*                                                                             *
*=============================================================================*

CONTENTS OF DISK

The disk contains 3 directories:

1:\USER :This directory contains 2 directories:
      \INCLUDE :
          I2C_PLM.H         :PLM header file
          I2C_C.H           :C header file
          I2C_MAC.DEF       :ASM header file,
                             Macro definitions for ASM function calls
          I2C_DATA.GLO      :I2C global data (assembler only)
          I2C_DATA.LOC      :I2C local data (assembler only, not for user)
          I2C_CODE.GLO      :I2C extern code definitions (assembler only)
          REG751.H          :8xC751 register file

      \LIB :
          LIB.BAT           :example batch file to create library
          I2C_751.LIB       :8xC751/2 I2C drive library

2:\EXAMPLE :This directory contains 3 directories
          \DEMO_ASM         :Assembly example
          \DEMO_PLM         :PL/M example
          \DEMO_C           :C example

3:\SOURCE  :This directory contains the source files of the modules that are
            put in library with I2C_751.LIB
```

I²C Master routines

```
$ TITLE(I2C_DEF.ASM)
;*=========================================================*
;*                                                         *
;*         INCLUDE FILE:  I2C_DEF.ASM                      *
;*         PACKAGE     :  I2C                              *
;*                                                         *
;*=========================================================*

;*---------------------------------------------------------*
;*    This file must be LINKED to each I2C sub function    *
;*---------------------------------------------------------*

    USING(1)

;*=========================================================*
;*    G L O B A L   D A T A   D E C L A R A T I O N S      *
;*=========================================================*
    PUBLIC      I2C_MCB
    PUBLIC      I2C_CTRL
    PUBLIC      I2C_STAT
    PUBLIC      OWN_SLV_ADDR
    PUBLIC      SLV_BUF_PTR

    PUBLIC      I2C_ADDR_1
    PUBLIC      BUF_LEN_1
    PUBLIC      BUF_PTR_1
    PUBLIC      I2C_ADDR_2
    PUBLIC      BUF_LEN_2
    PUBLIC      BUF_PTR_2


;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E C L A R A T I O N *
;*=========================================================*
    PUBLIC      _I2C_INIT_BYTE
    PUBLIC      _I2C_TEST_DEVICE_BYTE
    PUBLIC      _I2C_WRITE_BYTE
    PUBLIC      _I2C_WRITE_SUB_BYTE
    PUBLIC      _I2C_WRITE_SUB_SWINC_BYTE
    PUBLIC      _I2C_WRITE_MEMORY_BYTE
    PUBLIC      _I2C_WRITE_SUB_WRITE_BYTE
    PUBLIC      _I2C_WRITE_SUB_READ_BYTE
    PUBLIC      _I2C_WRITE_COM_WRITE_BYTE
    PUBLIC      _I2C_WRITE_REP_WRITE_BYTE
    PUBLIC      _I2C_WRITE_REP_READ_BYTE
    PUBLIC      _I2C_READ_BYTE
    PUBLIC      _I2C_READ_STATUS_BYTE
    PUBLIC      _I2C_READ_SUB_BYTE
    PUBLIC      _I2C_READ_REP_READ_BYTE
    PUBLIC      _I2C_READ_REP_WRITE_BYTE


;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E F I N I T I O N S *
;*=========================================================*
    I2C_MCB_DATA   SEGMENT DATA
    RSEG           I2C_MCB_DATA

    _I2C_INIT_BYTE:
        OWN_SLV_ADDR:   DS      1
        SLV_BUF_PTR:    DS      1


    _I2C_TEST_DEVICE_BYTE:
    _I2C_WRITE_BYTE:
    _I2C_WRITE_SUB_BYTE:
    _I2C_WRITE_SUB_SWINC_BYTE:
    _I2C_WRITE_MEMORY_BYTE:
    _I2C_WRITE_SUB_WRITE_BYTE:
    _I2C_WRITE_SUB_READ_BYTE:
    _I2C_WRITE_COM_WRITE_BYTE:
    _I2C_WRITE_REP_WRITE_BYTE:
    _I2C_WRITE_REP_READ_BYTE:
    _I2C_READ_BYTE:
    _I2C_READ_STATUS_BYTE:
    _I2C_READ_SUB_BYTE:
    _I2C_READ_REP_READ_BYTE:
    _I2C_READ_REP_WRITE_BYTE:
```

```
        I2C_MCB:        DS      6
            I2C_ADDR_1          DATA    I2C_MCB+0
            BUF_LEN_1           DATA    I2C_MCB+1
            BUF_PTR_1           DATA    I2C_MCB+2
            I2C_ADDR_2          DATA    I2C_MCB+3
            BUF_LEN_2           DATA    I2C_MCB+4
            BUF_PTR_2           DATA    I2C_MCB+5


    I2C_STAT_DATA  SEGMENT DATA BITADDRESSABLE
    RSEG           I2C_STAT_DATA

    I2C_CTRL:           DS      1
    I2C_STAT:           DS      1

END
```

```
$ TITLE(I2CDATAG.H)
;*=========================================================*
;*                                                         *
;*         INCLUDE FILE:  I2CDATA.GLO                      *
;*         PACKAGE     :  I2C                              *
;*                                                         *
;*=========================================================*

;*---------------------------------------------------------*
;*    This file must be included into each I2C function,   *
;*    and into the MAIN ASSEMBLER program (if exists)      *
;*    It contains the I2C Global data definitions          *
;*---------------------------------------------------------*



;*=========================================================*
;*     I 2 C   F R E Q U E N C Y   S E T T I N G S         *
;*=========================================================*

;*---------------------------------------------------------*
;*     This part contains frequency dependent settings     *
;*       of the 8xC751/8xC752 I2C interface                *
;*                                                         *
;*     The user can adapt this part to his own wishes.     *
;*     If this part has been changed, the whole I2C package *
;*       must be assembled, linked and put into a library  *
;*       again.                                            *
;*                                                         *
;*     The default setting are made for a 16MHz clock      *
;*       frequency, and a 40 mS delay for programming EEPROM *
;*---------------------------------------------------------*
CT1_CT0         EQU         002H   ;Frequency <= 16.8  MHz
                         ;  001H    Frequency <= 14.25 MHz
                         ;  000H    Frequency <= 11.7  MHz
                         ;  003H    Frequency <=  9.14 MHz


EEPROM_PROG_DELAY   EQU     103    ;40 Msec at 16MHz

               l DELAY =   514 * EEPROM_PROG_DELAY * 12/fosc
;*=========================================================*
;*     E N D   I 2 C   F R E Q U E N C Y   S E T T I N G S *
;*=========================================================*


;*=========================================================*
;*    G L O B A L   D A T A   D E F I N I T I O N S        *
;*=========================================================*
    EXTRN DATA (_I2C_INIT_BYTE)
        EXTRN DATA (OWN_SLV_ADDR)
        EXTRN DATA (SLV_BUF_PTR)

    EXTRN DATA (I2C_MCB)
        EXTRN DATA (I2C_ADDR_1)
        EXTRN DATA (BUF_LEN_1)
        EXTRN DATA (BUF_PTR_1)
        EXTRN DATA (I2C_ADDR_2)
        EXTRN DATA (BUF_LEN_2)
        EXTRN DATA (BUF_PTR_2)

    EXTRN DATA (I2C_CTRL)
        REP_STRT_BLK1   BIT     I2C_CTRL.0
        REP_BLK1        BIT     I2C_CTRL.1
        ADDR2           BIT     I2C_CTRL.2
        ADDR2_SUB       BIT     I2C_CTRL.3
        BLOCK2          BIT     I2C_CTRL.4
        RWN_BLK2        BIT     I2C_CTRL.5
        REP_STRT_BLK2   BIT     I2C_CTRL.6
        TEST_DEVICE     BIT     I2C_CTRL.7

    EXTRN DATA (I2C_STAT)
        RETRY_0         BIT     I2C_STAT.0
        RETRY_1         BIT     I2C_STAT.1
        RETRY_2         BIT     I2C_STAT.2
        I2C_ERR         BIT     I2C_STAT.3
        TIME_ERR        BIT     I2C_STAT.4
        RECOVER         BIT     I2C_STAT.5
        BUS_RECOVERED   BIT     I2C_STAT.6
        NO_ACK          BIT     I2C_STAT.7
```

```
;*=========================================================*
;*    G L O B A L   S Y M B O L   D E C L A R A T I O N S   *
;*=========================================================*
    I2C_START_CTRL          EQU     0D0H+CT1_CT0
    I2C_ENABLE              EQU     080H+CT1_CT0
    I2C_RELEASE             EQU     0F4H

    C_XMTA                  EQU     080H
    C_IDLE                  EQU     040H
    C_DRDY                  EQU     020H
    C_ARL                   EQU     010H
    C_STRT                  EQU     008H
    C_STP                   EQU     004H
    S_RSTR                  EQU     022H
    S_STP                   EQU     021H
```

```
$ TITLE(I2C_DATA.LOC)
;*=========================================================*
;*                                                         *
;*         INCLUDE FILE:  I2C_DATA.LOC                      *
;*         PACKAGE     :  I2C                              *
;*                                                         *
;*=========================================================*

;*---------------------------------------------------------*
;*    This file must be included into each I2C function,   *
;*    It contains the I2C Local symbol definitions         *
;*---------------------------------------------------------*

;*=========================================================*
;*      L O C A L   S Y M B O L   D E F I N I T I O N S    *
;*=========================================================*
    SDA             BIT         81H
    SCL             BIT         80H

    BUF_PTR         SET         R0
    BUF_LEN         SET         R1
    BIT_CNT         SET         R2
    MESS_RETRY_CNT  SET         R3
    BUS_ERR_CLKS    SET         R4
    MEM_MESS_LEN    SET         R5
    MEM_DELAY_H     SET         R6
    MEM_DELAY_L     SET         R7
```

```
$ TITLE(I2C_CODE.H)
;*=========================================================*
;*                                                         *
;*         INCLUDE FILE:  I2C_CODE.H                       *
;*         PACKAGE     :  I2C                              *
;*                                                         *
;*=========================================================*

;*---------------------------------------------------------*
;*    This file must be included into the ASSEMBLER MAIN   *
;*    It contains the EXTERNAL CODE references (Global     *
;*    function definitions) of the I2C functions           *
;*---------------------------------------------------------*

;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E F I N I T I O N S *
;*=========================================================*
    EXTRN CODE(_I2C_INIT)
    EXTRN CODE(_I2C_TEST_DEVICE)
    EXTRN CODE(_I2C_WRITE)
    EXTRN CODE(_I2C_WRITE_SUB)
    EXTRN CODE(_I2C_WRITE_SUB_SWINC)
    EXTRN CODE(_I2C_WRITE_MEMORY)
    EXTRN CODE(_I2C_WRITE_SUB_WRITE)
    EXTRN CODE(_I2C_WRITE_SUB_READ)
    EXTRN CODE(_I2C_WRITE_COM_WRITE)
    EXTRN CODE(_I2C_WRITE_REP_WRITE)
    EXTRN CODE(_I2C_WRITE_REP_READ)
    EXTRN CODE(_I2C_READ)
    EXTRN CODE(_I2C_READ_STATUS)
    EXTRN CODE(_I2C_READ_SUB)
    EXTRN CODE(_I2C_READ_REP_READ)
    EXTRN CODE(_I2C_READ_REP_WRITE)
```

```
$ TITLE(I2C_Init command)
;*=========================================================*
;*                                                         *
;*         SOURCE FILE :  I2C_INIT.ASM                     *
;*         PACKAGE     :  I2C                              *
;*                                                         *
;*=========================================================*
$DEBUG


;*=========================================================*
;*    I N C L U D E S                                      *
;*=========================================================*


;*=========================================================*
;*    L O C A L   S Y M B O L   D E C L A R A T I O N S    *
;*=========================================================*
    RETRIES         SET     R3

$NOLIST
$INCLUDE(REG751.H)
$INCLUDE(I2C_DATA.GLO)
$INCLUDE(I2C_DATA.LOC)
$LIST


;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E F I N I T I O N S  *
;*=========================================================*
  PUBLIC _I2C_INIT

;*=========================================================*
;*    C O D E   S E G M E N T                              *
;*=========================================================*
    I2C_DRIVER SEGMENT CODE
    RSEG I2C_DRIVER

;*MPF:::I2C::I2C_INIT.ASM:I2C_INIT============================*
;*                                                         *
;* FUNCTION NAME:      I2C_INIT                            *
;*PACKAGE:             I2C                                 *
;* DESCRIPTION:                                            *
;*  Initialize I2C interface: set SDA & SCL, enable time out *
;*  timer, allow 16 MHz (CT1,CT0 = 0). Set the number of     *
;*  retries (max 7) into the I2C_STAT. Bit 7,6 and 5 of the  *
;*  I2C_STAT contain the number of retries. Those bits may   *
;*  not be changed during the I2C routines.                 *
;*                                                         *
;* INPUT:                                                  *
;*   Before calling I2C_INIT the main program must take care *
;*   that the correct parameters are available in          *
;*   OWN_SLV_ADDR, SLV_BUF_PTR and _I2C_INIT_BYTE+2, this    *
;*   is done automatically when using C, PL/M or the pre-    *
;*   defined assembler macro (available in I2C_MAC.DEF)     *
;*                                                         *
;* OUTPUT:                                                 *
;*   initialized I2C and retry number in I2C_STAT 7..5       *
;*                                                         *
;*EMP==========================================================*
_I2C_INIT:
       MOV        I2CFG,#I2C_ENABLE     ;CLR TIRUN, CLR MASTRQ
       SETB       ETI
       SETB       EI2
       SETB       EA                    ;enable interrupts
       ANL        OWN_SLV_ADDR,#0FEH    ;save slv addr bit 0=0
       MOV        I2C_STAT,_I2C_INIT_BYTE+2
       ANL        I2C_STAT,#07H         ;I2C_STAT = retries
       MOV        I2CON,#I2C_RELEASE
       RET


;*=========================================================*
;*    H I S T O R Y                                        *
;*=========================================================*
;*                                                         *
;* 03-07-91    J.C. Pijnenburg  original version           *
;*                                                         *
;*=========================================================*
       END
```

```
$TITLE(I2C age Handler)
;*=========================================================*
;*                                                         *
;*         SOURCE FILE : I2C_HAND.ASM                      *
;*         PACKAGE     : I2C                               *
;*                                                         *
;*=========================================================*
$DEBUG


;*=========================================================*
;*    I N C L U D E S                                      *
;*=========================================================*
$NOLIST
$INCLUDE(REG751.H)
$INCLUDE(I2C_DATA.GLO)
$INCLUDE(I2C_DATA.LOC)
$LIST


;*=========================================================*
;*    G L O B A L   R E F E R E N C E S                    *
;*=========================================================*
    EXTRN CODE(I2C_STOP)
    EXTRN CODE(I2C_TRX_BYTE)
    EXTRN CODE(I2C_TRX_ADDR)
    EXTRN CODE(I2C_RCV_BYTE)
    EXTRN CODE(I2C_TRX_BLOCK)
    EXTRN CODE(I2C_RCV_BLOCK)
    EXTRN CODE(I2C_STRT_SLVAD)
    EXTRN CODE(I2C_RSTRT_SLVAD)

;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E F I N I T I O N S  *
;*=========================================================*
    PUBLIC I2C_MESS_HAND


;*=========================================================*
;*    L O C A L   S Y M B O L   D E C L A R A T I O N S    *
;*=========================================================*
    RWN         BIT     0E0H        ;bit ACC.0
    I2C_PSW     EQU     8


;*=========================================================*
;*    C O D E   S E G M E N T                              *
;*=========================================================*
    I2C_DRIVER SEGMENT CODE
    RSEG I2C_DRIVER

;*MPF:::I2C::I2C_HAND.ASM:I2C_MESS_HAND=====================*
;*                                                         *
;* FUNCTION NAME:      I2C_HAND                            *
;* PACKAGE:            I2C                                 *
;* DESCRIPTION:                                            *
;*   Transmit an I2C age, includes error handling          *
;*                                                         *
;* INPUT: age control byte I2C_CTRL (bit addressable)      *
;*        age control block I2C_MCB, containing:           *
;*          I2C_ADDR1       (i.e. slave address            *
;*          BUF_LEN1        (i.e. number of bytes to trx.) *
;*          BUF_PTR1        (i.e. transmit buffer)         *
;*          I2C_ADDR2       (i.e. sub address)             *
;*          BUF_LEN2        (i.e. length of second data blk) *
;*          BUF_PTR2        (i.e. second transmit buffer)  *
;*                                                         *
;* OUTPUT: I2C_ERROR byte  (bit addressable)               *
;*                                                         *
;*EMP========================================================*
I2C_MESS_HAND:
        PUSH    PSW
        MOV     PSW,#I2C_PSW        ;sel RB1
        ANL     I2C_STAT,#07H       ;clr all but retry bits
        MOV     MESS_RETRY_CNT,I2C_STAT
        INC     MESS_RETRY_CNT      ;load retry counter
```

```
RETRY:
        ANL     I2C_STAT,#07H       ;clr all but retry bits
        MOV     A,I2C_ADDR_1        ;load SLV_ADDR
        CLR     RWN                 ;if (subaddress)
        JB      ADDR2_SUB,STRT      ;    RWN = 0
        MOV     C,RWN_BLK1          ;else
        MOV     RWN,C               ;    RWN = RWN_BLK1
STRT:
        ACALL   I2C_STRT_SLVAD      ;send START+SLV_ADDR+RWN
        JNB     I2C_ERR,CONTINUE    ;branch offset to large
        AJMP    EXIT
CONTINUE:
        JB      TEST_DEVICE,M_STOP
        MOV     BUF_PTR,BUF_PTR_1   ;load pointer block1
        MOV     BUF_LEN,BUF_LEN_1   ;load length blodk1
        JNB     ADDR2_SUB,BLOCK     ;if (addr2_sub)
        MOV     A,I2C_ADDR_2        ;  load sub address
        ACALL   I2C_TRX_BYTE        ;  trx_byte(sub address0
        JB      I2C_ERR,EXIT        ;  if (error) exit ();
        JNB     REP_STRT_BLK1,BLOCK ;  if (rep. start blk1)
        MOV     A,I2C_ADDR_1        ;    load slave address
        SETB    RWN                 ;    read
        ACALL   I2C_RSTRT_SLVAD     ;    send RSTART+SLV_ADDR
        JB      I2C_ERR,EXIT        ;    if (error) exit();
BLOCK:
        JNB     RWN_BLK1,TRX_1      ;if ((rwn_blk1) == read))
        ACALL   I2C_RCV_BLOCK       ;  rcv_block(&data1,cnt1)
        SJMP    END_BLOCK1
TRX_1:
        ACALL   I2C_TRX_BLOCK       ;  trx_block(&dtat1,cnt1)
ENC_BLOCK1:
        JB      I2C_ERR,EXIT        ;if (error) exit();
        JNB     BLOCK2,M_STOP       ;if (2nd block of data)
        MOV     BUF_PTR,BUF_PTR_2   ;{
        MOV     BUF_LEN,BUF_LEN_2   ;
        JNB     ADDR2,DATA2         ;  if (addr2)
        JNB     REP_STRT_BLK2,DATA2 ;    if (rep. start blk2)
        MOV     A,I2C_ADDR_2        ;    { set address2
        JNB     ADDR2_SUB,SET_RWN   ;      if(addr2_sub)
        MOV     A,I2C_ADDR_1        ;        set address1
SET_RWN:                            ;        /* same slave */
        MOV     C,RWN_BLK2
        MOV     RWN,C               ;        modify RWN BLK1
        ACALL   I2C_RSTRT_SLVAD     ;      snd RSTART+SLV_ADDR
        JB      I2C_ERR,EXIT        ;      if (error) exit();
DATA2:                              ;    }
        JNB     RWN_BLK2,TRX_2      ;  if ((rwn_blk2)==read)
        ACALL   I2C_RCV_BLOCK       ;    rcv_block(&_dat1,c1)
        SJMP    BLOCK_ERR
TRX_2:                              ;  else
        ACALL   I2C_TRX_BLOCK       ;    trx_block(&data1,c1)
BLOCK_ERR:
        JB      I2C_ERR,EXIT        ;    if (error exit();
M_STOP:
        ACALL   I2C_STOP            ;}
        JB      I2C_ERR,EXIT        ;    if (error exit();
        AJMP    RESTORE_CONTEXT


;*MPF:::I2C::I2C_HAND.ASM:EXIT==============================*
;*                                                          *
;* FUNCTION NAME:      EXIT                                 *
;* PACKAGE:            I2C                                  *
;* DESCRIPTION:                                             *
;*   Exit an I2C message. This routine is only entered if an*
;*   I2C error has occured. If more retries must be made,   *
;*   the message is started again. If no retry must be made,*
;*   the message handler is left after setting the carry.   *
;*   Carry is 1 indicates that an error has occurred (return*
;*   value for C and PL/M calls). If the routine is entered *
;*   at the RESTORE_CONTEXT label, no error has occurred    *
;*                                                          *
;* OUTPUT: I2C_ERROR byte  (bit addressable                 *
;*                                                          *
;*EMP=========================================================*
EXIT:
        JNB     TIME_ERR,TO_RETRY
        JNB     BUS_RECOVERED,RESTORE_CONTEXT
```

```
TO_RETRY:
        MOV     I2CON,#I2C_RELEASE
        DJNZ    MESS_RETRY_CNT,RETRY ; if (no more retries)

RESTORE_CONTEXT:
        MOV     I2CFG,#I2C_ENABLE   ; SLVEN=1,MSTRQ=0,TIRN=0
        MOV     A,I2C_STAT
        POP     PSW                 ;   restore PSW
        MOV     C,I2C_ERR
END_MESSAGE:                        ;label for debugging
        SETB    EI2                 ;   enable I2C interrupt
        RET                         ; with XRAY

;*=========================================================*
;*     H I S T O R Y                                       *
;*=========================================================*
;*                                                         *
;* 12-06-91    J.C. Pijnenburg  original version           *
;*                                                         *
;*=========================================================*
        END
```

```
$TITLE(I2C_Basic Functions)
;*=========================================================*
;*                                                         *
;*         SOURCE FILE : I2C_BASI.ASM                      *
;*         PACKAGE     : I2C                               *
;*                                                         *
;*=========================================================*
$DEBUG


;*=========================================================*
;*    I N C L U D E S                                      *
;*=========================================================*
$NOLIST
$INCLUDE(REG751.H)
$INCLUDE(I2C_DATA.GLO)
$INCLUDE(I2C_DATA.LOC)
$LIST


;*=========================================================*
;*    G L O B A L   R E F E R E N C E S                    *
;*=========================================================*
    EXTRN CODE(I2C_SLV_TRX)
    EXTRN CODE(I2C_SLV_RCV)
    EXTRN CODE(ADDR_RECOG)


;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E F I N I T I O N S *
;*=========================================================*
    PUBLIC I2C_STRT_SLVAD
    PUBLIC I2C_RSTRT_SLVAD
    PUBLIC I2C_STOP
    PUBLIC I2C_TRX_BYTE
    PUBLIC I2C_TRX_ADDR
    PUBLIC I2C_RCV_BYTE
    PUBLIC I2C_RCV_ADDR
    PUBLIC I2C_TRX_BLOCK
    PUBLIC I2C_RCV_BLOCK
    PUBLIC ADDR_COMPARE


;*=========================================================*
;*    I N T E R R U P T   C O D E   S E G M :   T I M E R 1 *
;*=========================================================*
    CSEG AT 01BH


;*MPF:::I2C::I2C_INIT.ASM:I2C_TIME_OUT=====================*
;*                                                         *
;* FUNCTION NAME:      I2C_TIME_OUT                        *
;* PACKAGE:            I2C                                 *
;* DESCRIPTION:                                            *
;*  I2C time out routine, clear timer interrupt, set the   *
;*  TIME_ERR bit. If the 8xC751 was I2C bus master while the*
;*  interrupt occurred (RECOVER = 1), and attempt to recover*
;*  the bus is made.                                       *
;*  To recover, SDA and SCL are set, if SCL remains low, the*
;*  I2C bus cannot be recovered and the routine is left.   *
;*  If SCL is HIGH but SDA is low, 9 additional clocks are *
;*  genrated. If SDA becomes HIGH, a STOP is made          *
;*  If the 8xC751 was not I2C bus master (RECOVER = 0), the *
;*  bus is released.                                       *
;*                                                         *
;*EMP=========================================================*
       SETB    CLRTI
       SETB    TIME_ERR
       CLR     TIRUN
       AJMP    TI_INT


;*=========================================================*
;*    C O D E   S E G M E N T                              *
;*=========================================================*
    I2C_DRIVER SEGMENT CODE
    RSEG I2C_DRIVER
```

```
;*MPP:::I2C::I2C_BASI=========================================*
;*                                                           *
;* SOURCE FILE:        I2C_BASI.ASM                          *
;* PACKAGE:            I2C                                   *
;* DESCRIPTION:                                              *
;*  Generate a start condition on the I2C bus, Set the       *
;*  MASTRQ bit. If 8xC751 has not become master on ATN,      *
;*  switch to receive mode and check if the own slave        *
;*  address is received.                                     *
;*                                                           *
;* INPUT: none                                               *
;* OUTPUT: I2C_ERROR (0, no error; 1, error)                 *
;* OUTPUT CONDITION: SCL is stretched                        *
;*                                                           *
;*EMP=========================================================*
I2C_STRT_SLVAD;
        SETB    TIRUN
        JB      STR,IS_MASTER       ;already started
        CLR     EI2                 ;disable I2C interrupt
        MOV     I2CFG,#I2C_START_CTRL
        ACALL   WAIT_ATN
IS_MASTER:
        JB      MASTER,I2C_TRX_ADDR
        MOV     I2CCON,#C_STRT
        ACALL   I2C_RCV_ADDR
        JB      I2C_ERR,END_I2C_START
        ACALL   ADDR_COMPARE
START_ERR:
        SETB    I2C_ERR
END_I2C_START:
        RET


;*MPF:::I2C::I2C_BASI.ASM:I2C_REP_STRT=======================*
;*                                                           *
;* FUNCTION NAME:      I2C_REP_START                         *
;* PACKAGE:            I2C                                   *
;* DESCRIPTION:                                              *
;*  Generate a repeated start condition on the I2C bus       *
;*  The repeated start is generated by setting the XSTR      *
;*  bit. If STR is not set (by hardware), the I2C bus is     *
;*  released, no check for own slave address is done after a*
;*  repeated start.                                          *
;*                                                           *
;* INPUT: none                                               *
;* OUTPUT: I2C_ERROR (0, no error; 1, error)                 *
;* OUTPUT CONDITION: SCL is stretched                        *
;*                                                           *
;*EMP=========================================================*
I2C_RSTRT_SLVAD:
        MOV     I2CON,#S_RSTR
        ACALL   WAIT_ATN                ;wait for rising SCL
        JNB     DRDY,I2C_BASIC_ERR
        MOV     I2CON,#C_DRDY
        ACALL   WAIT_ATN                ;wait for rep start
        JNB     STR,I2C_BASIC_ERR
        SJMP    I2C_TRX_ADDR


;*MPF:::I2C::I2C_BASI.ASM:I2C_STOP===========================*
;*                                                           *
;* FUNCTION NAME:      I2C_STOP                              *
;* PACKAGE             I2C                                   *
;* DESCRIPTION:                                              *
;*  Generate a stop condition on the I2C bus                 *
;*  The STOP condition is generated by setting the XSTP bit.*
;*  If no error occurs, this function is left with I2C bus   *
;*  released and TI stopped. In case of an error the bus is  *
;*  released in the message handler.                         *
;*                                                           *
;* INPUT: none                                               *
;* OUTPUT: I2C_ERROR (0, no error; 1, error)                 *
;*                                                           *
;*EMP=========================================================*
I2C_STOP:
        CLR     MASTRQ
        MOV     I2CCON,#S_STP
        ACALL   WAIT_ATN                ;wait for rising SCL
        JNB     DRDY,I2C_BASIC_ERR
        MOV     I2CON,#C_DRDY
        ACALL   WAIT_ATN                ;wait for stop
```

```
        MOV     I2CON,#I2C_RELEASE
        CLR     TIRUN
        RET


;*MPF:::I2C::I2C_BASI.ASM:I2C_TRX_ADDR=====================*
;*                                                         *
;* FUNCTION NAME:     I2C_TRX_ADDR                         *
;* PACKAGE:           I2C                                  *
;* DESCRIPTION:                                            *
;*  This function calls I2C_TRX_BYTE to transmit the       *
;*  slave address, if an arbitration is lost before the last*
;*  bit is transmitted, the function receives the remaining *
;*  bits (receive mode), and checks whether the own slave  *
;*  address has been received (call ADDR_CMP).             *
;*                                                         *
;* INPUT byte to transmit in ACC                           *
;* OUTPUT: I2C_ERROR (0, no error; 1, error)               *
;* OUTPUT CONDITION: SCL is stretched                      *
;*                                                         *
;*EMP=======================================================*
I2C_TRX_ADDR:
        ACALL   I2C_TRX_BYTE
        JNB     I2C_ERR,END_TRX_ADDR
        DJNZ    BIT_CNT,CONTINUE
        AJMP    END_TRX_ADDR
CONTINUE:
        JNB     ARL,END_TRX_ADDR
        JB      STR,END_TRX_ADDR        ;parasitaire START
        JB      STR,END_TRX_ADDR        ;parasitaire STOP
        CLR     I2C_ERR                 ;clr err for slv func
        INC     BIT_CNT                 ;correct BIT_CNT
        CLR     0E0H                    ;RDAT = 0 to ACC.0
RCV_NEXT_BIT:
        MOV     I2CON,#C_XMTA+C_DRDY+C_ARL  ;rcv mode
        ACALL   WAIT_ATN
        JNB     DRDY,RESTORE_ERR
        MOV     C,RDAT
        RLC     A
        DJNZ    BIT_CNT,RCV_NEXT_BIT
        ACALL   ADDR_COMPARE
RESTORE_ERR:
        SETB    I2C_ERR                 ;set err for ret main
END_TRX_ADDR:
        RET


;*MPF:::I2C::I2C_BASI.ASM:I2C_ADDR_COMPARE==================*
;*                                                         *
;* FUNCTION NAME:      I2C_ADDR_COMPARE                    *
;* PACKAGE:            I2C                                 *
;* DESCRIPTION:                                            *
;*  Compares the contents of the accumulator (received     *
;*  address) with the OWN_SLV_ADDR. If equal and the RWN   *
;*  bit is 0 (master transmit, slave receive) I2C_CLV_RCV is*
;*  called. If equal and the RWN bit is 1 (master receive, *
;*  slave transmit) I2C_SLV_TRX is called. If not euqal exit*
;*  I2C_ADDR_COMPARE                                       *
;*                                                         *
;* INPUT: received address in ACC                          *
;* OUTPUT: I2C_ERROR (0, no error; 1, error)               *
;* OUTPUT CONDITION: SCL is stretched                      *
;*                                                         *
;*EMP=======================================================*
ADDR_COMPARE:
        XRL     A,OWN_SLV_ADDR
        JZ      SEND_ACK
        CJNE    A,#1,END_ADDR_COMPARE
SEND_ACK:
        MOV     I2DAT,#0
        ACALL   WAIT_ATN
        JNB     DRDY,END_ADDR_COMPARE
        JZ      SLAVE_RCV
        AJMP    I2C_SLV_TRX
SLAVE_RCV:
        AJMP    I2C_SLV_RCV
END_ADDR_COMPARE:
        RET
```

```
;*MPF:::I2C::I2C_BASI.ASM:I2C_BASIC_ERR=====================*
;*                                                          *
;* FUNCTION NAME:          I2C_BASIC_ERR                    *
;* PACKAGE:                I2C                              *
;* DESCRIPTION:                                             *
;*  Set the I2C_ERR bit. The message handler tests this bit *
;*                                                          *
;* INPUT: none                                              *
;* OUTPUT: I2C_ERROR = 1                                    *
;*                                                          *
;*EMP=========================================================*
I2C_BASIC_ERR:
        SETB    I2C_ERR
        RET


;*MPF:::I2C::I2C_BASI.ASM:I2C_TRX_BYTE======================*
;*                                                          *
;* FUNCTION NAME:      I2C_TRX_BYTE                         *
;* PACKAGE:            I2C                                  *
;* DESCRIPTION:                                             *
;*  Transmit a byte over the I2C bus.                       *
;*  NOTE: The STR bit is cleared here instead of in the     *
;*        I2C_START routine, because there must be valid    *
;*        data in I2DAT before STR may be cleared (also     *
;*        releases the SCL line).                           *
;*                                                          *
;*  The I2C_TRX_BYTE function transmits a byte over the I2C *
;*  bus, after the last bit has been transmitted,           *
;*  the function switches to receive mode to receive the    *
;*  acknowledge bit. If NACK is received, the NO_ACK bit is *
;*  set. If arbitration is lost or an error occurs during   *
;*  I2C_TRX_BYTE the function is exit with the I2C_ERR bit  *
;*  set.                                                    *
;* INPUT: byte to transmit in ACC                           *
;* OUTPUT: I2C ERROR (0, no error; 1, error)                *
;* OUTPUT CONDITION: SCL is stretched                       *
;*                                                          *
;*EMP=========================================================*
I2C_TRX_BYTE:
        MOV     BIT_CNT,#8
TRX_BIT:
        MOV     I2DAT,A             ;release SCL
        MOV     I2CON,#C_STRT       ;if STR clear STR
                                            ;else dummy MOV
        ACALL   WAIT_ATN
        JNB     DRDY,I2C_BASIC_ERR
        RL      A
        DJNZ    BIT_CNT,TRX_BIT
        MOV     I2CON,#C_XMTA+C_DRDY    ;receive mode
        ACALL   WAIT_ATN
        JNB     DRDY,I2C_BASIC_ERR
        JNB     RDAT,TRX_BYTE_RDY   ;stretch SCL
        SETB    NO_ACK
TRX_BYTE_RDY:
        RET


;*MPF:::I2C::I2C_BASI.ASM:I2C_RCV_BYTE======================*
;*                                                          *
;* FUNCTION NAME:      I2C_RCV_BYTE                         *
;* PACKAGE:            I2C                                  *
;* DESCRIPTION:                                             *
;*  Receive a byte from te I2C bus                          *
;*  This is one function which receives a byte into acc.    *
;*  RCV_BYTE first releases the SCL and then receives the 8 *
;*  bits. If RCV_ADDR is called, the first bit is already in*
;*  the RDAT register, this must first be saved before the  *
;*  SCL is released.                                        *
;*                                                          *
;* INPUT: none                                              *
;* OUTPUT: I2C_ERROR (0, no error; 1, error)                *
;*         if (! I2C_ERROR) received byte in ACC.           *
;*                                                          *
;*EMP=========================================================*
I2C_RCV_BYTE:
        MOV     I2CON,#C_XMTA+C_DRDY     ;rel. SCL, rcv mode
I2C_RCV_ADDR:
        MOV     BIT_CNT,#8
        CLR     A                   ;  rcv first bit
```

```
RCV_BIT:
        ACALL   WAIT_ATN
        JNB     DRDY,I2C_BASIC_ERR
        DJNZ    BIT_CNT,NOT_LAST_BIT
        MOV     C,RDAT
        RLC     A
        RET
NOT_LAST_BIT:
        ORL     A,I2DAT             ; save bit, rel. SCL
        RL      A
        SJMP    RCV_BIT


;*MPF:::I2C::I2C_BASI.ASM:I2C_TRX_BLOCK=====================*
;*                                                          *
;* FUNCTION NAME:      I2C_TRX_BLOCK                        *
;* PACKAGE:            I2C                                  *
;* DESCRIPTION:                                             *
;*  Transmit a block of bytes over the I2C bus              *
;*  The I2C_TRX_BLOCK function transmits as much bytes as   *
;*  defined in BUF_LEN (set R2), before the message handler *
;*  is called, BUF_LEN_1 or BUF_LEN_2 is copied into BUF_LEN*
;*                                                          *
;* INPUT: pointer to begin of block data in BUF_PTR (R0)    *
;*        byte counter in BIF_LEN (R2)                      *
;* OUTPUT: I2C_ERROR (0, no error; 1, error)                *
;*                                                          *
;*EMP=======================================================*
I2C_TRX_BLOCK:
        MOV     A,@BUF_PTR          ;load byte
        ACALL   I2C_TRX_BYTE
        JB      I2C_ERR,END_TRX_BLOCK
        JB      NO_ACK,END_TRX_BLOCK
        INC     BUF_PTR
        DJNZ    BUF_LEN,I2C_TRX_BLOCK
END_TRX_BLOCK:
        RET


;*MPF:::I2C::I2C_BASI.ASM:I2C_RCV_BLOCK=====================*
;*                                                          *
;* FUNCTION NAME:      I2C_RCV_BLOCK                        *
;* PACKAGE:            I2C                                  *
;* DESCRIPTION:                                             *
;*  Receive a block of bytes from the I2C bus               *
;*  The I2C_RCV_BLOCK function receives as much bytes as    *
;*  defined in BUF_LEN (set R2), before the message handler *
;*  is called, BUF_LEN_1 or BUF_LEN_2 is copied into BUF_LEN*
;*                                                          *
;* INPUT: pointer to begin of receive buffer BUF_PTR (R0)   *
;*        byte counter in BUF_LEN (R2)                      *
;* OUTPUT: I2C_ERROR (0, no error; 1, error)                *
;*         if (!I2C_ERROR)   received bytes in buffer.      *
;*                                                          *
;*EMP=======================================================*
ACK_RCV_BYTE:
        MOV     I2DAT,#0            ;send ACK
        ACALL   WAIT_ATN
        JNB     DRDY,I2C_BASIC_ERR
I2C_RCV_BLOCK:
        ACALL   I2C_RCV_BYTE
        JB      I2C_ERR,END_RCV_BLOCK
        MOV     @BUF_PTR,A          ;save byte
        INC     BUF_PTR
        DJNZ    BUF_LEN,ACK_RCV_BYTE
        MOV     I2DAT,#80H          ;send NACK
        ACALL   WAIT_ATN
        JNB     DRDY,I2C_BASIC_ERR
END_RCV_BLOCK:
        RET


;*MPF:::I2C::I2C_BASI.ASM:WAIT_ATN==========================*
;*                                                          *
;* FUNCTION NAME:      WAIT_ATN                             *
;* PACKAGE:            I2C                                  *
;* DESCRIPTION:                                             *
;*   The WAIT_ATN function waits for the ATN bit to be set. *
;*   The function is left if the ATN bit is set or if the   *
;*   TIME_ERR bit is set. The TIME_ERR bit indicates that a *
```

```
;*   bus timeout has occurred. If the 8xC751 enters this    *
;*   function as a master, the RECOVER bit is set,          *
;*   indicating that in case of a timeout, a bus recover    *
;*   action must be started.                                *
;*                                                          *
;*EMP=========================================================*
WAIT_ATN:
         JNB     MASTER,WAIT
         SETB    RECOVER
WIAT:
         JB      TIME_ERR,END_WAIT
         JNB     ATN,WAIT
END_WAIT:
         CLR     RECOVER
         RET


;*MPF:::I2C::I2C_BASI.ASM:TI_INT==============================*
;*                                                          *
;* FUNCTION NAME:      TI_INT                               *
;* PACKAGE:            I2C                                  *
;* DESCRIPTION:                                             *
;*   The TI_INT handles the timeout interrupt. It is        *
;*   entered when a time out occurs (during wAIT_ATN).      *
;*   The function is placed here to be sure that it is      *
;*   linked when placed in a library.                       *
;*                                                          *
;*EMP=========================================================*
TI_INT:
         CLR     092H
         ACALL   SCL_DELAY
         SETB    092H
         JNB     RECOVER,RET_INT
         SETB    SDA
         SETB    SCL
         JNB     SCL,RET_INT
         MOV     BUS_ERR_CLKS,#9         ;SCL = 1

RETRY_LOOP:
         CLR     SCL
         ACALL   SCL_DELAY
         SETB    SCL
         ACALL   SCL_DELAY
         JB      SDA,MAKE_STOP        ;if SDA = 1 make stop
         DJNZ    BUS_ERR_CLKS,RETRY_LOOP
         RETI
MAKE_STOP:
         CLR     SCL
         NOP
         CLR     SDA
         ACALL   SCL_DELAY
         SETB    SCL
         ACALL   SCL_DELAY
         SETB    SDA                  ;make stop condition
         ACALL   SCL_DELAY
         SETB    BUS_RECOVERED
RET_INT:
         RETI

SCL_DELAY:               ;delay of 9 periods (>= 6 micro sec.)
         NOP             ; ACALL(2) + 5 NOP (4) + RET (2)
         NOP
         NOP
         NOP
         NOP
         RET

;*=========================================================*
;*     H I S T O R Y                                       *
;*=========================================================*
;*                                                         *
;* 03-07-91    J.C. Pijnenburg  original version           *
;*                                                         *
;*=========================================================*
        END
```

```
$TITLE(I2C_Test_Device command)
;*=========================================================*
;*                                                         *
;*         SOURCE FILE : I2C_TDEV.ASM                      *
;*         PACKAGE     : I2C                               *
;*                                                         *
;*=========================================================*
$DEBUG


;*=========================================================*
;*    I N C L U D E S                                      *
;*=========================================================*
$NOLIST
$INCLUDE (I2C_DATA.GLO)
$LIST


;*=========================================================*
;*    G L O B A L   R E F E R E N C E S                    *
;*=========================================================*
    EXTRN CODE(I2C_MESS_HAND)


;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E F I N I T I O N S *
;*=========================================================*
    PUBLIC _I2C_TEST_DEVICE


;*=========================================================*
;*    L O C A L   S Y M B O L   D E C L A R A T I O N S    *
;*=========================================================*
    TEST_DEVICE_MASK  EQU 80H
                      ;REP_STRT_BLK1 = 0     (NO)
                      ;RWN_BLK1      = 0     (WRITE)
                      ;ADDR2         = 0     (NO)
                      ;ADDR2_SUB     = 0     (--)
                      ;BLOCK2        = 0     (NO)
                      ;RWN_BLK2      = 0     (--)
                      ;REP_STRT_BLK2 = 0     (--)
                      ;T_DEVICE      = 1     (--)


;*=========================================================*
;*    C O D E   S E G M E N T                              *
;*=========================================================*
    I2C_DRIVER SEGMENT CODE
    RSEG I2C_DRIVER


;*MPF:::I2C::I2C_TDEV.ASM:I2C_TEST_DEVICE===================*
;*                                                         *
;* FUNCTION NAME:      I2C_TEST_DEVICE                     *
;* PACKAGE:            I2C                                 *
;* DESCRIPTION:                                            *
;*   Address a slave , if ack received slave was present   *
;*                                                         *
;* PROTOCOL:                                               *
;*   <S><SLV_ADDR><W><A><P>                                *
;*                                                         *
;* INPUT: Message control byte I2C_CTRL (bit addressable)  *
;*        Message control block I2C_MCB, containing:       *
;*          I2C_ADDR1       (slave address)                *
;* OUTPUT: I2C_ERROR byte   (bit addressable)              *
;*                                                         *
;*EMP========================================================*

_I2C_TEST_DEVICE:
        MOV     I2C_CTRL,#TEST_DEVICE_MASK
        AJMP    I2C_MESS_HAND

        END
```

```
$TITLE(I2C_Write command)
;*=========================================================*
;*                                                         *
;*         SOURCE FILE : I2C_WRIT.ASM                      *
;*         PACKAGE     : I2C                               *
;*                                                         *
;*=========================================================*
$DEBUG


;*=========================================================*
;*    I N C L U D E S                                      *
;*=========================================================*
$NOLIST
$INCLUDE (I2C_DATA.GLO)
$LIST


;*=========================================================*
;*    G L O B A L   R E F E R E N C E S                    *
;*=========================================================*
    EXTRN CODE(I2C_MESS_HAND)


;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E F I N I T I O N S *
;*=========================================================*
    PUBLIC _I2C_WRITE


;*=========================================================*
;*    L O C A L   S Y M B O L   D E C L A R A T I O N S    *
;*=========================================================*
    WRITE_MASK  EQU 00H
                    ;REP_STRT_BLK1 = 0     (NO)
                    ;RWN_BLK1      = 0     (WRITE)
                    ;ADDR2         = 0     (NO)
                    ;ADDR2_SUB     = 0     (--)
                    ;BLOCK2        = 0     (NO)
                    ;RWN_BLK2      = 0     (--)
                    ;REP_STRT_BLK2 = 0     (--)
                    ;TEST_DEVICE   = 0     (--)


;*=========================================================*
;*    C O D E   S E G M E N T                              *
;*=========================================================*
    I2C_DRIVER SEGMENT CODE
    RSEG I2C_DRIVER


;*MPF:::I2C::I2C_WRIT.ASM:I2C_WRITE=======================*
;*                                                         *
;* FUNCTION NAME:      I2C_WRITE                           *
;* PACKAGE:            I2C                                 *
;* DESCRIPTION:                                            *
;*   Write n bytes to a slave device.                      *
;*                                                         *
;* PROTOCOL:                                               *
;*   <S><SLV_ADDR><W><A><D0><A><D1><A>. .<Dn-1><A><P>      *
;*                                                         *
;* INPUT: Message control byte I2C_CTRL (bit addressable)  *
;*        Message control block I2C_MCB, containing:       *
;*          I2C_ADDR1       (slave address)                *
;*          BUF_LEN1        (number of bytes (n) to trx.)  *
;*          BUF_PTR1        (ptr to transmit buffer)       *
;*                                                         *
;* OUTPUT: I2C_ERROR byte   (bit addressable)              *
;*                                                         *
;*EMP======================================================*

_I2C_WRITE:
        MOV     I2C_CTRL,#WRITE_MASK
        AJMP    I2C_MESS_HAND

        END
```

```
$TITLE(I2C_Write_Sub command)
;*=========================================================*
;*                                                         *
;*         SOURCE FILE : I2C_WSUB.ASM                      *
;*         PACKAGE     : I2C                               *
;*                                                         *
;*=========================================================*
$DEBUG


;*=========================================================*
;*    I N C L U D E S                                      *
;*=========================================================*
$NOLIST
$INCLUDE (I2C_DATA.GLO)
$LIST


;*=========================================================*
;*    G L O B A L   R E F E R E N C E S                    *
;*=========================================================*
    EXTRN CODE(I2C_MESS_HAND)


;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E F I N I T I O N S *
;*=========================================================*
    PUBLIC _I2C_WRITE_SUB


;*=========================================================*
;*    L O C A L   S Y M B O L   D E C L A R A T I O N S    *
;*=========================================================*
   WRITE_SUB_MASK  EQU 00H
                    ;REP_STRT_BLK1 = 0     (NO)
                    ;RWN_BLK1      = 0     (WRITE)
                    ;ADDR2         = 1     (YES)
                    ;ADDR2_SUB     = 1     (YES)
                    ;BLOCK2        = 0     (NO)
                    ;RWN_BLK2      = 0     (--)
                    ;REP_STRT_BLK2 = 0     (--)
                    ;TEST_DEVICE   = 0     (--)


;*=========================================================*
;*    C O D E   S E G M E N T                              *
;*=========================================================*
    I2C_DRIVER SEGMENT CODE
    RSEG I2C_DRIVER


;*MPF:::I2C::I2C_WSUB.ASM:I2C_WRITE_SUB=====================*
;*                                                         *
;* FUNCTION NAME:      I2C_WRITE_SUB                       *
;* PACKAGE:            I2C                                 *
;* DESCRIPTION:                                            *
;*   Write a block of data (a length n) preceded by a      *
;*   sub address to a slave device.                        *
;*                                                         *
;* PROTOCOL:                                               *
;*   <S><SLV_ADDR><W><A><SUB_ADDR><A><Da0><A><Da1><A>..<A> *
;*                                          <Dan-1><A><P>  *
;*                                                         *
;* INPUT: Message control byte I2C_CTRL (bit addressable)  *
;*        Message control block I2C_MCB, containing:       *
;*          I2C_ADDR1       (slave address)                *
;*          BUF_LEN1        (number of bytes in block )    *
;*          BUF_PTR1        (ptr to block )                *
;*          I2C_ADDR2       (sub address)                  *
;*                                                         *
;* OUTPUT: I2C_ERROR byte   (bit addressable)              *
;*                                                         *
;*EMP========================================================*

_I2C_WRITE_SUB:
        MOV     I2C_CTRL,#WRITE_SUB_MASK
        AJMP    I2C_MESS_HAND

        END
```

```
$TITLE(I2C_Write_Sub_SWinc command)
;*=========================================================*
;*                                                         *
;*         SOURCE FILE : I2C_WSWI.ASM                      *
;*         PACKAGE     : I2C                               *
;*                                                         *
;*=========================================================*
$DEBUG


;*=========================================================*
;*    I N C L U D E S                                      *
;*=========================================================*
$NOLIST
$INCLUDE(I2C_DATA.GLO)
$INCLUDE(I2C_DATA.LOC)
$INCLUDE(REG751.H)
$LIST


;*=========================================================*
;*    G L O B A L   R E F E R E N C E S                    *
;*=========================================================*
    EXTRN CODE(I2C_MESS_HAND)


;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E F I N I T I O N S *
;*=========================================================*
    PUBLIC _I2C_WRITE_SUB_SWINC
    PUBLIC _I2C_WRITE_MEMORY


;*=========================================================*
;*    L O C A L   S Y M B O L   D E C L A R A T I O N S    *
;*=========================================================*
    WRITE_SUB_SWINC_MASK  EQU 00CH
    WRITE_MEMORY_MASK     EQU 02CH
                    ;REP_STRT_BLK1  = 0     (NO)
                    ;RWN_BLK1       = 0     (WRITE)
                    ;ADDR2          = 1     (YES)
                    ;ADDR2_SUB      = 1     (YES)
                    ;BLOCK2         = 0     (NO)
                    ;RWN_BLK2       = 0 or 1 (no blk2,
                                        used for delay/no delay)
                    ;REP_STRT_BLK2  = 0     (--)
                    ;TEST_DEVICE    = 0     (--)


;*=========================================================*
;*    C O D E   S E G M E N T                              *
;*=========================================================*
    I2C_DRIVER SEGMENT CODE
    RSEG I2C_DRIVER


;*MPF:::I2C::I2C_WSWI.ASM:I2C_WRITE_SUB_SWINC===============*
;*                                                         *
;* FUNCTION NAME:      I2C_WRITE_SUB_SWINC                 *
;* PACKAGE:            I2C                                 *
;* DESCRIPTION:                                            *
;*   Transmit an I2C message, the message is split into    *
;*   sub messages. Each sub message transmits one byte.    *
;*   If the slave is an EEPROM a delay is generated after  *
;*   each sub message. The RWN_BLK2 is not used in the     *
;*   message handler (no block 2) and is therefore free to *
;*   distinguish between write to EEPROM (1=delay) and other*
;*   (0= no delay)
;*                                                         *
;* PROTOCOL:                                               *
;*   <S><SLV_ADDR><W><A><SUB_ADDR><A><D0><A><P>            *
;*        if (RWN_BLK2) delay 40 ms                        *
;*   <S><SLV_ADDR><W><A><SUB_ADDR+1><A><D1><A><P>          *
;*        if (RWN_BLK2) delay 40 ms                        *
;*   <-    --      -  - --   ->                            *
;*        if (RWN_BLK2) delay 40 ms                        *
;*   <S><SLV_ADDR><W><A><SUB_ADDR+n-1><A><Dn-1><P>         *
;*        if (RWN_BLK2) delay 40 ms                        *
;*                                                         *
```

```
;* INPUT: Message control byte I2C_CTRL (bit addressable)   *
;*        Message control block I2C_MCB, containing:        *
;*          I2C_ADDR1       (slave address)                 *
;*          BUF_LEN1        (number of bytes (mess) to trx.)*
;*          BUF_PTR1        (ptr to transmit buffer)        *
;*          I2C_ADDR2       (sub address)                   *
;*                                                          *
;* OUTPUT: I2C_ERROR byte   (bit addressable)               *
;*                                                          *
;*EMP=========================================================*

_I2C_WRITE_MEMORY:
       MOV      I2C_CTRL,#WRITE_MEMORY_MASK
       SJMP     SET_MESS_CNT
_I2C_WRITE_SUB_SWINC:
       MOV      I2C_CTRL,#WRTIE_SUB_SWINC_MASK

SET_MESS_CNT:
       MOV      MEM_MESS_LEN,I2C_MCB+1
       MOV      I2C_MCB+1,#1            ;set BUF_LEN_1 = 1
SUB_MESS;
       ACALL    I2C_MESS_HAND
       JB       I2C_ERR,END_WSWI
       INC      I2C_MCB+2               ;inc. BUFF_PTR_1
       INC      I2C_MCB+3               ;inc. SUB_ADDR
       JNB      RWN_BLK2,NEXT
       MOV      MEM_DELAY_H,#EEPROM_PROG_DELAY
       MOV      MEM_DELAY_L,#00         ; 40 mS delay at 16MHz
PROGRAM_DELAY:
       DJNZ     MEM_DELAY_L,$
       DJNZ     MEM_DELAY_H,PROGRAM_DELAY
NEXT:
       DJNZ     MEM_MESS_LEN,SUB-MESS
END_WSWI:
       RET

       END
```

```
$TITLE(I2C_Write_Sub_Write command)
;*=========================================================*
;*                                                         *
;*         SOURCE FILE : I2C_WSUW.ASM                      *
;*         PACKAGE     : I2C                               *
;*                                                         *
;*=========================================================*
$DEBUG


;*=========================================================*
;*    I N C L U D E S                                      *
;*=========================================================*
$NOLIST
$INCLUDE(I2C_DATA.GLO)
$LIST


;*=========================================================*
;*    G L O B A L   R E F E R E N C E S                    *
;*=========================================================*
    EXTRN CODE(I2C_MESS_HAND)


;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E F I N I T I O N S *
;*=========================================================*
    PUBLIC _I2C_WRITE_SUB_WRITE


;*=========================================================*
;*    L O C A L   S Y M B O L   D E C L A R A T I O N S    *
;*=========================================================*
    WRITE_SUB_WRITE_MASK  EQU 1CH
                      ;REP_STRT_BLK1  = 0     (NO)
                      ;RWN_BLK1       = 0     (WRITE)
                      ;ADDR2          = 1     (YES)
                      ;ADDR2_SUB      = 1     (YES)
                      ;BLOCK2         = 1     (YES)
                      ;RWN_BLK2       = 0     (wRITE)
                      ;REP_STRT_BLK2  = 0     (NO)
                      ;TEST_DEVICE    = 0     (--)


;*=========================================================*
;*    C O D E   S E G M E N T                              *
;*=========================================================*
    I2C_DRIVER SEGMENT CODE
    RSEG I2C_DRIVER


;*MPF:::I2C::I2C_WSUW.ASM:I2C_WRITE_SUB_WRITE===============*
;*                                                         *
;* FUNCTION NAME:      I2C_WRITE_SUB_WRITE                 *
;* PACKAGE:            I2C                                 *
;* DESCRIPTION:                                            *
;*   Write 2 blocks of data (a and b, length n and m)      *
;*   preceded by a sub address into a single slave device  *
;*                                                         *
;* PROTOCOL:                                               *
;*   <S><SLV_ADDR><W><A><SUB_ADDR><A><Da0><A>..<A><Dan-1><A>*
;*                                  <Db0><A>..<Dbm-1><A><P>*
;*                                                         *
;* INPUT: Message control byte I2C_CTRL (bit addressable)  *
;*        Message control block I2C_MCB, containing:       *
;*          I2C_ADDR1       (slave address)                *
;*          BUF_LEN1        (number of bytes in block a)   *
;*          BUF_PTR1        (ptr to block a)               *
;*          I2C_ADDR2       (sub address)                  *
;*          BUF_LEN1        (number of bytes in block b)   *
;*          BUF_PTR1        (ptr to block b)               *
;*                                                         *
;* OUTPUT: I2C_ERROR byte   (bit addressable)              *
;*                                                         *
;*EMP=======================================================*

_I2C_WRITE_SUB_WRITE:
        MOV     I2C_CTRL,#WRITE_SUB_WRITE_MASK
        AJMP    I2C_MESS_HAND

        END
```

```
$TITLE(I2C_Write_Sub_Read command)
;*=========================================================*
;*                                                         *
;*         SOURCE FILE : I2C_WSUR.ASM                      *
;*         PACKAGE     : I2C                               *
;*                                                         *
;*=========================================================*
$DEBUG


;*=========================================================*
;*    I N C L U D E S                                      *
;*=========================================================*
$NOLIST
$INCLUDE(I2C_DATA.GLO)
$LIST


;*=========================================================*
;*    G L O B A L   R E F E R E N C E S                    *
;*=========================================================*
    EXTRN CODE(I2C_MESS_HAND)


;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E F I N I T I O N S *
;*=========================================================*
    PUBLIC _I2C_WRITE_SUB_READ


;*=========================================================*
;*    L O C A L   S Y M B O L   D E C L A R A T I O N S    *
;*=========================================================*
    WRITE_SUB_READ_MASK  EQU 7CH
                     ;REP_STRT_BLK1  = 0     (NO)
                     ;RWN_BLK1       = 0     (WRITE)
                     ;ADDR2          = 1     (YES)
                     ;ADDR2_SUB      = 1     (YES)
                     ;BLOCK2         = 1     (YES)
                     ;RWN_BLK2       = 1     (READ)
                     ;REP_STRT_BLK2  = 1     (YES)
                     ;TEST_DEVICE    = 0     (--)


;*=========================================================*
;*    C O D E   S E G M E N T                              *
;*=========================================================*
    I2C_DRIVER SEGMENT CODE
    RSEG I2C_DRIVER


;*MPF:::I2C::I2C_WSUR.ASM:I2C_WRITE_SUB_READ================*
;*                                                         *
;* FUNCTION NAME:      I2C_WRITE_SUB_READ                  *
;* PACKAGE:            I2C                                 *
;* DESCRIPTION:                                            *
;*   Write a block of data (a length n) preceded by a sub  *
;*   address, generate repeated start and read a second    *
;*   block of data (b length m) from the slave device      *
;*                                                         *
;* PROTOCOL:                                               *
;*   <S><SLV_ADDR><W><A><SUB_ADDR><A><Da0><A>..<A><DaN-1><A>*
;*   <S><SLV_ADDR><R><A><Db0><A>..<A><Dbm-1><N><P>         *
;*                                                         *
;* INPUT: Message control byte I2C_CTRL (bit addressable)  *
;*        Message control block I2C_MCB, containing:       *
;*          I2C_ADDR1       (slave address)                *
;*          BUF_LEN1        (number of bytes in block a)   *
;*          BUF_PTR1        (ptr to block a)               *
;*          I2C_ADDR2       (sub address)                  *
;*          BUF_LEN1        (number of bytes in block b)   *
;*          BUF_PTR1        (ptr to block b)               *
;*                                                         *
;* OUTPUT: I2C_ERROR byte   (bit addressable)              *
;*                                                         *
;*EMP========================================================*

_I2C_WRITE_SUB_READ:
        MOV     I2C_CTRL,#WRITE_SUB_READ_MASK
        AJMP    I2C_MESS_HAND

        END
```

```
$TITLE(I2C_Write_Com_Write command)
;*=========================================================*
;*                                                         *
;*         SOURCE FILE : I2C_WCOW.ASM                      *
;*         PACKAGE     : I2C                               *
;*                                                         *
;*=========================================================*
$DEBUG


;*=========================================================*
;*    I N C L U D E S                                      *
;*=========================================================*
$NOLIST
$INCLUDE(I2C_DATA.GLO)
$LIST


;*=========================================================*
;*    G L O B A L   R E F E R E N C E S                    *
;*=========================================================*
    EXTRN CODE(I2C_MESS_HAND)


;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E F I N I T I O N S *
;*=========================================================*
    PUBLIC _I2C_WRITE_COM_WRITE


;*=========================================================*
;*    L O C A L   S Y M B O L   D E C L A R A T I O N S    *
;*=========================================================*
    WRITE_COM_WRITE_MASK  EQU 10H
                      ;REP_STRT_BLK1  = 0     (NO)
                      ;RWN_BLK1       = 0     (WRITE)
                      ;ADDR2          = 0     (NO)
                      ;ADDR2_SUB      = 0     (--)
                      ;BLOCK2         = 1     (YES)
                      ;RWN_BLK2       = 0     (WRITE)
                      ;REP_STRT_BLK2  = 0     (--)
                      ;TEST_DEVICE    = 0     (NO)


;*=========================================================*
;*    C O D E   S E G M E N T                              *
;*=========================================================*
    I2C_DRIVER SEGMENT CODE
    RSEG I2C_DRIVER


;*MPF:::I2C::I2C_WCOW.ASM:I2C_WRITE_COM_WRITE===============*
;*                                                         *
;* FUNCTION NAME:      I2C_WRITE_COM_WRITE                 *
;* PACKAGE:            I2C                                 *
;* DESCRIPTION:                                            *
;*   Write a 2 blocks of data (a,b length n,m) in a single  *
;*   message to a slave device                             *
;*   another slave device.                                 *
;*                                                         *
;* PROTOCOL:                                               *
;*   <S><SLV_ADDR1><W><A><Da0><A><Da1><A>..<A><Dan-1><A>    *
;*                        <Db0><A><Db1><A>..<A><Dbm-1><A><P> *
;*                                                         *
;* INPUT: Message control byte I2C_CTRL (bit addressable)   *
;*        Message control block I2C_MCB, containing:        *
;*          I2C_ADDR1       (slave address first device)    *
;*          BUF_LEN1        (number of bytes in block a)    *
;*          BUF_PTR1        (ptr to block a)                *
;*          BUF_LEN1        (number of bytes in block b)    *
;*          BUF_PTR1        (ptr to block b)                *
;*                                                         *
;* OUTPUT: I2C_ERROR byte   (bit addressable)               *
;*                                                         *
;*EMP=======================================================*

_I2C_WRITE_COM_WRITE:
        MOV     I2C_MCB+5,I2C_MCB+4
        MOV     I2C_MCB+4,I2C_MCB+3
        MOV     I2C_CTRL,#WRITE_COM_WRITE_MASK
        AJMP    I2C_MESS_HAND

        END
```

```
$TITLE(I2C_Write_Rep_Write command)
;*=========================================================*
;*                                                         *
;*         SOURCE FILE : I2C_WREW.ASM                      *
;*         PACKAGE     : I2C                               *
;*                                                         *
;*=========================================================*
$DEBUG


;*=========================================================*
;*    I N C L U D E S                                      *
;*=========================================================*
$NOLIST
$INCLUDE(I2C_DATA.GLO)
$LIST


;*=========================================================*
;*    G L O B A L   R E F E R E N C E S                    *
;*=========================================================*
    EXTRN CODE(I2C_MESS_HAND)


;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E F I N I T I O N S *
;*=========================================================*
    PUBLIC _I2C_WRITE_REP_WRITE


;*=========================================================*
;*    L O C A L   S Y M B O L   D E C L A R A T I O N S    *
;*=========================================================*
    WRITE_REP_WRITE_MASK  EQU 54H
                     ;REP_STRT_BLK1  = 0     (NO)
                     ;RWN_BLK1       = 0     (WRITE)
                     ;ADDR2          = 1     (YES)
                     ;ADDR2_SUB      = 0     (NO)
                     ;BLOCK2         = 1     (YES)
                     ;RWN_BLK2       = 0     (WRITE)
                     ;REP_STRT_BLK2  = 1     (YES)
                     ;TEST_DEVICE    = 0     (--)


;*=========================================================*
;*    C O D E   S E G M E N T                              *
;*=========================================================*
    I2C_DRIVER SEGMENT CODE
    RSEG I2C_DRIVER


;*MPF:::I2C::I2C_WREW.ASM:I2C_WRITE_REP_WRITE===============*
;*                                                         *
;* FUNCTION NAME:      I2C_WRITE_REP_WRITE                 *
;* PACKAGE:            I2C                                 *
;* DESCRIPTION:                                            *
;*   Write a block of data (a length n) to a slave device, *
;*   sent repeated start and write a block (b length m) to *
;*   another slave device.                                 *
;*                                                         *
;* PROTOCOL:                                               *
;*   <S><SLV_ADDR1><W><A><Da0><A><Da1><A>..<A><Dan-1><A>   *
;*   <S><SLV_ADDR2><W><A><Db0><A><Db1><A>..<A><Dbm-1><N><P>*
;*                                                         *
;* INPUT: Message control byte I2C_CTRL (bit addressable)  *
;*        Message control block I2C_MCB, containing:       *
;*          I2C_ADDR1       (slave address first device)   *
;*          BUF_LEN1        (number of bytes in block a)   *
;*          BUF_PTR1        (ptr to block a)               *
;*          I2C_ADDR2       (slave address second device)  *
;*          BUF_LEN1        (number of bytes in block b)   *
;*          BUF_PTR1        (ptr to block b)               *
;*                                                         *
;* OUTPUT: I2C_ERROR byte   (bit addressable)              *
;*                                                         *
;*EMP========================================================*

_I2C_WRITE_REP_WRITE:
        MOV     I2C_CTRL,#WRITE_REP_WRITE_MASK
        AJMP    I2C_MESS_HAND

        END
```

```
$TITLE(I2C_Write_Rep_Read command)
;*=========================================================*
;*                                                         *
;*         SOURCE FILE : I2C_WRER.ASM                      *
;*         PACKAGE     : I2C                               *
;*                                                         *
;*=========================================================*
$DEBUG


;*=========================================================*
;*    I N C L U D E S                                      *
;*=========================================================*
$NOLIST
$INCLUDE(I2C_DATA.GLO)
$LIST


;*=========================================================*
;*    G L O B A L   R E F E R E N C E S                    *
;*=========================================================*
    EXTRN CODE(I2C_MESS_HAND)


;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E F I N I T I O N S *
;*=========================================================*
    PUBLIC _I2C_WRITE_REP_READ


;*=========================================================*
;*    L O C A L   S Y M B O L   D E C L A R A T I O N S    *
;*=========================================================*
    WRITE_REP_READ_MASK  EQU 74H
                     ;REP_STRT_BLK1  = 0     (NO)
                     ;RWN_BLK1       = 0     (WRITE)
                     ;ADDR2          = 1     (YES)
                     ;ADDR2_SUB      = 0     (NO)
                     ;BLOCK2         = 1     (YES)
                     ;RWN_BLK2       = 1     (READ)
                     ;REP_STRT_BLK2  = 1     (YES)
                     ;TEST_DEVICE    = 0     (--)


;*=========================================================*
;*    C O D E   S E G M E N T                              *
;*=========================================================*
    I2C_DRIVER SEGMENT CODE
    RSEG I2C_DRIVER


;*MPF:::I2C::I2C_WRER.ASM:I2C_WRITE_REP_READ================*
;*                                                         *
;* FUNCTION NAME:      I2C_WRITE_REP_READ                  *
;* PACKAGE:            I2C                                 *
;* DESCRIPTION:                                            *
;*   Write a block of data (a length n) to a slave device, *
;*   sent repeated start and read a block (b length m) from *
;*   another slave device.                                 *
;*                                                         *
;* PROTOCOL:                                               *
;*   <S><SLV_ADDR1><W><A><Da0><A><Da1><A>..<A><Dan-1><A>   *
;*   <S><SLV_ADDR2><R><A><Db0><A><Db1><A>..<A><Dbm-1><N><P> *
;*                                                         *
;* INPUT: Message control byte I2C_CTRL (bit addressable)  *
;*        Message control block I2C_MCB, containing:       *
;*          I2C_ADDR1       (slave address first device)   *
;*          BUF_LEN1        (number of bytes in block a)   *
;*          BUF_PTR1        (ptr to block a)               *
;*          I2C_ADDR2       (slave address second device)  *
;*          BUF_LEN1        (number of bytes in block b)   *
;*          BUF_PTR1        (ptr to block b)               *
;*                                                         *
;* OUTPUT: I2C_ERROR byte   (bit addressable)              *
;*                                                         *
;*EMP========================================================*
_I2C_WRITE_REP_READ:
        MOV     I2C_CTRL,#WRITE_REP_READ_MASK
        AJMP    I2C_MESS_HAND

        END
```

```
$TITLE(I2C_Read command)
;*=========================================================*
;*                                                         *
;*         SOURCE FILE : I2C_READ.ASM                      *
;*         PACKAGE     : I2C                               *
;*                                                         *
;*=========================================================*
$DEBUG


;*=========================================================*
;*    I N C L U D E S                                      *
;*=========================================================*
$NOLIST
$INCLUDE(I2C_DATA.GLO)
$LIST


;*=========================================================*
;*    G L O B A L   R E F E R E N C E S                    *
;*=========================================================*
    EXTRN CODE(I2C_MESS_HAND)


;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E F I N I T I O N S *
;*=========================================================*
    PUBLIC _I2C_READ
    PUBLIC _I2C_READ_STATUS


;*=========================================================*
;*    L O C A L   S Y M B O L   D E C L A R A T I O N S    *
;*=========================================================*
    READ_MASK  EQU 02H
                    ;REP_STRT_BLK1  = 0     (NO)
                    ;RWN_BLK1       = 1     (READ)
                    ;ADDR2          = 0     (NO)
                    ;ADDR2_SUB      = 0     (--)
                    ;BLOCK2         = 0     (NO)
                    ;RWN_BLK2       = 0     (--)
                    ;REP_STRT_BLK2  = 0     (--)
                    ;TEST_DEVICE    = 0     (--)


;*=========================================================*
;*    C O D E   S E G M E N T                              *
;*=========================================================*
    I2C_DRIVER SEGMENT CODE
    RSEG I2C_DRIVER


;*MPF:::I2C::I2C_READ.ASM:I2C_READ=========================*
;*                                                         *
;* FUNCTION NAME:      I2C_READ                            *
;* PACKAGE:            I2C                                 *
;* DESCRIPTION:                                            *
;*   Read a block of data from a slave device (READ) or read*
;*   a single byte from a slave device (READ_STATUS)       *
;*                                                         *
;* PROTOCOL:                                               *
;*   <S><SLV_ADDR><R><A><D0><A><D1><A>..<A><Dn-1><N><P>    *
;*  or                                                     *
;*   <S><SLV_ADDR><R><A><STATUS><N><P>                     *
;*                                                         *
;* INPUT: Message control byte I2C_CTRL (bit addressable)  *
;*        Message control block I2C_MCB, containing:       *
;*          I2C_ADDR1       (slave address)                *
;*          BUF_LEN1        (number of bytes in block )    *
;*          BUF_PTR1        (ptr to store status)          *
;*                                                         *
;* OUTPUT: I2C_ERROR byte   (bit addressable)              *
;*                                                         *
;*EMP=======================================================*
_I2C_READ_STATUS:
        MOV     I2C_MCB+2,II2C_MCB+1
        MOV     I2C_MCB+1,#1         ;buffer length = 1
_i2C_READ:
        MOV     I2C_CTRL,#READ_MASK
        AJMP    I2C_MESS_HAND

        END
```

```
$TITLE(I2C_Read_Sub command)
;*=========================================================*
;*                                                         *
;*         SOURCE FILE : I2C_RSUB.ASM                      *
;*         PACKAGE     : I2C                               *
;*                                                         *
;*=========================================================*
$DEBUG


;*=========================================================*
;*    I N C L U D E S                                      *
;*=========================================================*
$NOLIST
$INCLUDE(I2C_DATA.GLO)
$LIST


;*=========================================================*
;*    G L O B A L   R E F E R E N C E S                    *
;*=========================================================*
    EXTRN CODE(I2C_MESS_HAND)


;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E F I N I T I O N S *
;*=========================================================*
    PUBLIC _I2C_READ_SUB


;*=========================================================*
;*    L O C A L   S Y M B O L   D E C L A R A T I O N S    *
;*=========================================================*
    READ_SUB_MASK  EQU 0FH
                   ;REP_STRT_BLK1  = 1     (YES)
                   ;RWN_BLK1       = 1     (READ)
                   ;ADDR2          = 1     (YES)
                   ;ADDR2_SUB      = 1     (YES)
                   ;BLOCK2         = 0     (NO)
                   ;RWN_BLK2       = 0     (--)
                   ;REP_STRT_BLK2  = 0     (--)
                   ;TEST_DEVICE    = 0     (--)


;*=========================================================*
;*    C O D E   S E G M E N T                              *
;*=========================================================*
    I2C_DRIVER SEGMENT CODE
    RSEG I2C_DRIVER


;*MPF:::I2C::I2C_RSUB.ASM:I2C_READ_SUB=====================*
;*                                                         *
;* FUNCTION NAME:      I2C_READ_SUB                        *
;* PACKAGE:            I2C                                 *
;* DESCRIPTION:                                            *
;*   Read a block of data (a length n) preceded by a       *
;*   sub address from a slave device.                      *
;*                                                         *
;* PROTOCOL:                                               *
;*   <S><SLV_ADDR><W><A><SUB_ADDR><A><S><SLV_ADDR><R><A>   *
;*                      <Da0><A><Da1><A>..<A><Dan-1><A><P> *
;*                                                         *
;* INPUT: Message control byte I2C_CTRL (bit addressable)  *
;*        Message control block I2C_MCB, containing:       *
;*          I2C_ADDR1       (slave address)                *
;*          BUF_LEN1        (number of bytes in block )    *
;*          BUF_PTR1        (ptr to block )                *
;*          I2C_ADDR2       (sub address)                  *
;*                                                         *
;* OUTPUT: I2C_ERROR byte   (bit addressable)              *
;*                                                         *
;*EMP======================================================*
_I2C_READ_SUB:
        MOV     I2C_CTRL,#READ_SUB_MASK
        AJMP    I2C_MESS_HAND

        END
```

```
$TITLE(I2C_Read_Rep_Write command)
;*=========================================================*
;*                                                         *
;*         SOURCE FILE : I2C_RREW.ASM                      *
;*         PACKAGE     : I2C                               *
;*                                                         *
;*=========================================================*
$DEBUG


;*=========================================================*
;*    I N C L U D E S                                      *
;*=========================================================*
$NOLIST
$INCLUDE(I2C_DATA.GLO)
$LIST


;*=========================================================*
;*    G L O B A L   R E F E R E N C E S                    *
;*=========================================================*
    EXTRN CODE(I2C_MESS_HAND)


;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E F I N I T I O N S *
;*=========================================================*
    PUBLIC _I2C_READ_REP_WRITE


;*=========================================================*
;*    L O C A L   S Y M B O L   D E C L A R A T I O N S    *
;*=========================================================*
    READ_REP_WRITE_MASK  EQU 56H
                    ;REP_STRT_BLK1  = 0     (NO)
                    ;RWN_BLK1       = 1     (READ)
                    ;ADDR2          = 1     (YES)
                    ;ADDR2_SUB      = 0     (NO)
                    ;BLOCK2         = 1     (YES)
                    ;RWN_BLK2       = 0     (WRITE)
                    ;REP_STRT_BLK2  = 1     (YES)
                    ;TEST_DEVICE    = 0     (--)


;*=========================================================*
;*    C O D E   S E G M E N T                              *
;*=========================================================*
    I2C_DRIVER SEGMENT CODE
    RSEG I2C_DRIVER


;*MPF:::I2C::I2C_RREW.ASM:I2C_READ_REP_WRITE================*
;*                                                         *
;* FUNCTION NAME:      I2C_READ_REP_WRITE                  *
;* PACKAGE:            I2C                                 *
;* DESCRIPTION:                                            *
;*   Read a block of data (a length n) from a slave device, *
;*   sent repeated start and write a block (b length m) to  *
;*   another slave device.                                 *
;*                                                         *
;* PROTOCOL:                                               *
;*   <S><SLV_ADDR1><R><A><Da0><A><Da1><A>..<A><Dan-1><N>    *
;*   <S><SLV_ADDR2><W><A><Db0><A><Db1><A>..<A><Dbm-1><A><P> *
;*                                                         *
;* INPUT: Message control byte I2C_CTRL (bit addressable)   *
;*        Message control block I2C_MCB, containing:        *
;*          I2C_ADDR1       (slave address first device)    *
;*          BUF_LEN1        (number of bytes in block a)    *
;*          BUF_PTR1        (ptr to block a)                *
;*          I2C_ADDR2       (slave address second device)   *
;*          BUF_LEN1        (number of bytes in block b)    *
;*          BUF_PTR1        (ptr to block b)                *
;*                                                         *
;* OUTPUT: I2C_ERROR byte   (bit addressable)               *
;*                                                         *
;*EMP========================================================*

_I2C_READ_REP_WRITE:
        MOV     I2C_CTRL,#READ_REP_WRITE_MASK
        AJMP    I2C_MESS_HAND

        END
```

```
$TITLE(I2C_Read_Rep_Read command)
;*=========================================================*
;*                                                         *
;*         SOURCE FILE : I2C_RRER.ASM                      *
;*         PACKAGE     : I2C                               *
;*                                                         *
;*=========================================================*
$DEBUG


;*=========================================================*
;*    I N C L U D E S                                      *
;*=========================================================*
$NOLIST
$INCLUDE(I2C_DATA.GLO)
$LIST


;*=========================================================*
;*    G L O B A L   R E F E R E N C E S                    *
;*=========================================================*
    EXTRN CODE(I2C_MESS_HAND)


;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E F I N I T I O N S *
;*=========================================================*
    PUBLIC _I2C_READ_REP_READ


;*=========================================================*
;*    L O C A L   S Y M B O L   D E C L A R A T I O N S    *
;*=========================================================*
    READ_REP_READ_MASK  EQU 076H
                    ;REP_STRT_BLK1  = 0     (NO)
                    ;RWN_BLK1       = 1     (READ)
                    ;ADDR2          = 1     (YES)
                    ;ADDR2_SUB      = 0     (NO)
                    ;BLOCK2         = 1     (YES)
                    ;RWN_BLK2       = 1     (READ)
                    ;REP_STRT_BLK2  = 1     (YES)
                    ;TEST_DEVICE    = 0     (--)


;*=========================================================*
;*    C O D E   S E G M E N T                              *
;*=========================================================*
    I2C_DRIVER SEGMENT CODE
    RSEG I2C_DRIVER


;*MPF:::I2C::I2C_RRER.ASM:I2C_READ_REP_READ================*
;*                                                         *
;* FUNCTION NAME:      I2C_READ_REP_READ                   *
;* PACKAGE:            I2C                                 *
;* DESCRIPTION:                                            *
;*   Read a block of data (a length n) from a slave device, *
;*   sent repeated start and read a block (b length m) from *
;*   another slave device.                                 *
;*                                                         *
;* PROTOCOL:                                               *
;*   <S><SLV_ADDR1><R><A><Da0><A><Da1><A>..<A><Dan-1><N>    *
;*   <S><SLV_ADDR2><R><A><Db0><A><Db1><A>..<A><Dbm-1><N><P> *
;*                                                         *
;* INPUT: Message control byte I2C_CTRL (bit addressable)  *
;*        Message control block I2C_MCB, containing:       *
;*          I2C_ADDR1       (slave address first device)   *
;*          BUF_LEN1        (number of bytes in block a)   *
;*          BUF_PTR1        (ptr to block a)               *
;*          I2C_ADDR2       (slave address second device)  *
;*          BUF_LEN1        (number of bytes in block b)   *
;*          BUF_PTR1        (ptr to block b)               *
;*                                                         *
;* OUTPUT: I2C_ERROR byte   (bit addressable)              *
;*                                                         *
;*EMP=======================================================*

_I2C_READ_REP_READ:
        MOV     I2C_CTRL,#READ_REP_READ_MASK
        AJMP    I2C_MESS_HAND

        END
```

I²C Slave routines

```
$TITLE(I2C_SLAVE)
;*=========================================================*
;*                                                         *
;*         SOURCE FILE : I2C_SLAV.ASM                      *
;*         PACKAGE     : I2C                               *
;*                                                         *
;*=========================================================*
$DEBUG


;*=========================================================*
;*    I N C L U D E S                                      *
;*=========================================================*
$NOLIST
$INCLUDE (I2C_DATA.GLO)
$INCLUDE(REG751.H)
$LIST


;*=========================================================*
;*    G L O B A L   R E F E R E N C E S                    *
;*=========================================================*
    EXTRN CODE(I2C_TRX_BYTE)
    EXTRN CODE(I2C_RCV_BYTE)
    EXTRN CODE(I2C_RCV_ADDR)
    EXTRN CODE(ADDR_COMPARE)


;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E F I N I T I O N S *
;*=========================================================*
    PUBLIC _I2C_SLV_TRX
    PUBLIC _I2C_SLV_RCV
    PUBLIC ADDR_RECOG


;*=========================================================*
;*    L O C A L   S Y M B O L   D E C L A R A T I O N S    *
;*=========================================================*
    SLV_BUF_PTR_W   SET         R1
    BIT_CNT         SET         R3
    I2C_RELEASE     EQU         0F4H


;*MPF:::I2C_SLAVE=========================================*
;*                                                         *
;* FUNCTION NAME:      I2C_SLAVE_ROUTINES                  *
;* DESCRIPTION:        I2C                                 *
;* DESCRIPTION:                                            *
;*    This file contains an example of how to make a slave  *
;*    transmitter and slave receiver function. The slave    *
;*    transmitter functions transmits byte from a buffer,   *
;*    while the slave receiver routine receives bytes into  *
;*    a buffer. The buffer pointer is loaded during the     *
;*    I2C_INIT routine.                                     *
;*                                                         *
;*EMP=========================================================*


;*=========================================================*
;*    I N T E R R U P T   C O D E   S E G M :   I I C      *
;*=========================================================*
    CSEG AT 023H

        PUSH    ACC
        PUSH    PSW
        MOV     PSW,#8
        AJMP    ADDR_RECOG


;*=========================================================*
;*    C O D E   S E G M E N T                              *
;*=========================================================*
    I2C_DRIVER SEGMENT CODE
    RSEG I2C_DRIVER
```

```
;*MPF:::I2C::I2C_SLAV.ASM:I2C_ADDR_RECOG====================*
;*                                                          *
;* FUNCTION NAME:      I2C_ADDR_RECOG                       *
;* PACKAGE NAME:       I2C                                  *
;* DESCRIPTION:                                             *
;*    If an I2C interrupt occurs, and the STR bit is set,   *
;*    I2C_ADDR_RECOG receives the incoming slave address.   *
;*    If its own slave address is recognized, the slave     *
;*    receiver or slave transmitter routine (depending on   *
;*    the R/WN bit) is called                               *
;*                                                          *
;* INPUT:  --                                               *
;* OUTPUT: I2C_ERROR byte  (bit addressable)                *
;*                                                          *
;*==========================================================*
ADDR_RECOG:
        MOV     I2CON,#C_STRT
        ACALL   I2C_RCV_ADDR
        JB      I2C_ERR,EXIT_SLV_AD
        ACALL   ADDR_COMPARE
EXIT_SLV_AD:
        CLR     I2C_ERR
        MOV     I2CON,#I2C_RELEASE
        POP     PSW
        POP     ACC
        RETI


;*MPF:::I2C::I2C_SLAV.ASM:I2C_SLV_TRX=======================*
;*                                                          *
;* FUNCTION NAME:      I2C_SLV_TRX                          *
;* PACKAGE NAME:       I2C                                  *
;* DESCRIPTION:                                             *
;*    After the SLV_ADDR/W is received, the I2C_SLV_TRX     *
;*    transmits a byte from the slave buffer. The pointer   *
;*    to this buffer is loaded during the I2C_INIT function.*
;*    If an acknowledge is received, the pointer is         *
;*    incremented and the next byte is transmitted. The     *
;*    function is exit on reception of a NACK.              *
;*                                                          *
;*    Normally the slave routines are entered through an I2C*
;*    interrupt, but if the 8xC751 loses arbitration during *
;*    the slave address and it recognizes its own slave     *
;*    address/W, the I2C_SLV_TRX function is entered at XXXX*
;*                                                          *
;* PROTOCOL: <S><SLV_ADDR><W><D0><A><D1><A>...<A><Dn><N><P> *
;*                                                          *
;* REGISTER USAGE : Register bank 1, is used during the I2C *
;*        routines, it contains no static data, and is free *
;*        for the user outside the I2C routines             *
;*                                                          *
;* INPUT:  --                                               *
;* OUTPUT: I2C_ERROR byte  (bit addressable)                *
;*                                                          *
;*EMP=======================================================*
I2C_SLV_TRX:
        MOV    SLV_BUF_PTR_W,SLV_BUF_PTR
S_TRX:
        MOV    A,@SLV_BUF_PTR_W
        ACALL  I2C_TRX_BYTE
        JB     I2C_ERR,EXIT_SLV_TRX
        JB     NO_ACK,EXIT_SLV_TRX
        INC    SLV_BUF_PTR_W
        AJMP   S_TRX
EXIT_SLV_TRX:
        RET
```

```
;*MPF:::I2C::I2C_SLAV.ASM:I2C_SLV_RCV=======================*
;*                                                          *
;* FUNCTION NAME:      I2C_SLV_RCV                          *
;* PACKAGE NAME:       I2C                                  *
;* DESCRIPTION:                                             *
;*   After the SLV_ADDR/R is received, the I2C_SLV_RCV      *
;*   receives a byte into the slave buffer. The pointer     *
;*   to this buffer is loaded during the I2C_INIT function. *
;*   After the byte is received, and acknowledge is send,   *
;*   the pointer is incremented and the next byte is        *
;*   received. The function is exit on if a start condition *
;*   is detected.                                           *
;*                                                          *
;*   Normally the slave routines are entered through and I2C*
;*   interrupt, but if the 8xC751 loses arbitration during  *
;*   the slave address and it recognizes its own slave      *
;*   address/R, the I2C_SLV_RCV function is entered at xx   *
;*                                                          *
;* PROTOCOL: <S><SLV_ADDR><R><D0><A><D1><A>..<A><Dn><A><P>  *
;*                                                          *
;* REGISTER USAGE : Register bank 1, is used during the I2C *
;*        routines, it contains no static data, and is free *
;*        fr the user outside the I2C routines              *
;*                                                          *
;* INPUT:  --                                               *
;* OUTPUT: I2C_ERROR byte  (bit addressable)                *
;*                                                          *
;*EMP=========================================================*
I2C-SLV_RCV:
        MOV    SLV_BUF_PTR_W,SLV_BUF_PTR
S_RCV:
        ACALL  I2C_RCV_BYTE
        JB     I2C_ERR,EXIT_SLV_RCV
        MOV    @SLV_BUF_PTR_WQ,A       ;save byte
        MOV    I2DAT,#0                ;send ACK
        JNB    ATN,$
        JNB    DRDY,EXIT_SLV_RCV
        INC    SLV_BUF_PTR_W
        AJMP   S_RCV
EXIT_SLV_RCV:
        RET


;*============================================================*
;*     H I S T O R Y                                          *
;*============================================================*
;*                                                          *
;* 21-05-91    J.C. Pijnenburg  original version            *
;*                                                          *
;*============================================================*
        END
```

I2C_MAC.DEF

```
#* DEFINE (I2C_INIT(Own_Slv_Addr,Slv_Buf_Addr,Retry))
   (
   MOV OWN_SLV_ADDR,#%Own_Slv_Addr
   MOV SLV_BUF_PTR, #%Slv_Buf_Addr
   MOV I2C_MCB,
   ACALL _I2C_INIT
   )

#* DEFINE (I2C_TEST_DEVICE(Slv_Addr))
   (
   MOV I2C_MCB,#%Slv_Addr
   ACALL _I2C_TEST_DEVICE
   )

#* DEFINE (I2C_WRITE(Slv_Addr,Count,Source_Ptr))
   (
   MOV I2C_MCB  ,#%Slv_Addr
   MOV I2C_MCB+1,#%Count
   MOV I2C_MCB+2,#%Source_Ptr
   ACALL _I2C_WRITE
   )

#* DEFINE (I2C_WRITE_SUB(Slv_Addr,Count,Source_Ptr,Sub_Addr))
   (
   MOV I2C_MCB  ,#%Slv_Addr
   MOV I2C_MCB+1,#%Count
   MOV I2C_MCB+2,#%Source_Ptr
   MOV I2C_MCB+3,#%Sub_Addr
   ACALL _I2C_WRITE_SUB
   )

#* DEFINE (I2C_WRITE_SUB_SWINC(Slv_Addr,Count,Source_Ptr,Sub_Addr))
   (
   MOV I2C_MCB  ,#%Slv_Addr
   MOV I2C_MCB+1,#%Count
   MOV I2C_MCB+2,#%Source_Ptr
   MOV I2C_MCB+3,#%Sub_Addr
   ACALL _I2C_WRITE_SUB_SWINC
   )

#* DEFINE (I2C_WRITE_MEMORY(Slv_Addr,Count,Source_Ptr,Sub_Addr))
   (
   MOV I2C_MCB  ,#%Slv_Addr
   MOV I2C_MCB+1,#%Count
   MOV I2C_MCB+2,#%Source_Ptr
   MOV I2C_MCB+3,#%Sub_Addr
   ACALL _I2C_WRITE_MEMORY
   )

#* DEFINE (I2C_WRITE_SUB_WRITE(Slv_Addr,Count_1,Source_Ptr_1,Sub_Addr,Count_2,Source_Ptr_2))
   (
   MOV I2C_MCB  ,#%Slv_Addr
   MOV I2C_MCB+1,#%Count_1
   MOV I2C_MCB+2,#%Source_Ptr_1
   MOV I2C_MCB+3,#%Sub_Addr
   MOV I2C_MCB+4,#%Count_2
   MOV I2C_MCB+5,#%Source_Ptr_2
   ACALL _I2C_WRITE_SUB_WRITE
   )

%* DEFINE (I2C_WRITE_SUB_READ(Slv_Addr,Count_1,Source_Ptr,Sub_Addr,Count_2,Dest_Ptr))
   (
   MOV I2C_MCB  ,#%Slv_Addr
   MOV I2C_MCB+1,#%Count_1
   MOV I2C_MCB+2,#%Source_Ptr
   MOV I2C_MCB+3,#%Sub_Addr
   MOV I2C_MCB+4,#%Count_2
   MOV I2C_MCB+5,#%Dest_Ptr
   ACALL _I2C_WRITE_SUB_READ
   )
```

```
%* DEFINE (I2C_WRITE_COM_WRITE(Slv_Addr,Count_1,Source_Ptr_1,Count_2,Source_Ptr_2))
   (
   MOV I2C_MCB  ,#%Slv_Addr
   MOV I2C_MCB+1,#%Count_1
   MOV I2C_MCB+2,#%Source_Ptr_1
   MOV I2C_MCB+3,#%Count_2
   MOV I2C_MCB+4,#%Source_Ptr_2
   ACALL _I2C_WRITE_COM_WRITE
   )

%* DEFINE (I2C_WRITE_REP_WRITE(Slv_Addr,Count_1,Source_Ptr_1,Sub_Addr,Count_2,Source_Ptr_2))
   (
   MOV I2C_MCB  ,#%Slv_Addr
   MOV I2C_MCB+1,#%Count_1
   MOV I2C_MCB+2,#%Source_Ptr_1
   MOV I2C_MCB+3,#%Sub_Addr
   MOV I2C_MCB+4,#%Count_2
   MOV I2C_MCB+5,#%Source_Ptr_2
   ACALL _I2C_WRITE_REP_WRITE
   )

%* DEFINE (I2C_WRITE_REP_READ(Slv_Addr,Count_1,Source_Ptr,Sub_Addr,Count_2,Dest_Ptr))
   (
   MOV I2C_MCB  ,#%Slv_Addr
   MOV I2C_MCB+1,#%Count_1
   MOV I2C_MCB+2,#%Source_Ptr
   MOV I2C_MCB+3,#%Sub_Addr
   MOV I2C_MCB+4,#%Count_2
   MOV I2C_MCB+5,#%Dest_Ptr
   ACALL _I2C_WRITE_REP_READ
   )

%* DEFINE (I2C_READ(Slv_Addr,Count,Dest_Ptr))
   (
   MOV I2C_MCB  ,#%Slv_Addr
   MOV I2C_MCB+1,#%Count
   MOV I2C_MCB+2,#%Dest_Ptr
   ACALL _I2C_READ
   )
%* DEFINE (I2C_READ_STATUS(Slv_Addr,Dest_Ptr))
   (
   MOV I2C_MCB  ,#%Slv_Addr
   MOV I2C_MCB+1,#%Dest_Ptr
   ACALL _I2C_READ_STATUS
   )

%* DEFINE (I2C_READ_SUB(Slv_Addr,Count,Dest_Ptr,Sub_Addr))
   (
   MOV I2C_MCB  ,#%Slv_Addr
   MOV I2C_MCB+1,#%Count
   MOV I2C_MCB+2,#%Dest_Ptr
   MOV I2C_MCB+3,#%Sub_Addr
   ACALL _I2C_READ_SUB
   )

%* DEFINE (I2C_READ_REP_READ(Slv_Addr,Count_1,Dest_Ptr_1,Sub_Addr,Count_2,Dest_Ptr_2))
   (
   MOV I2C_MCB  ,#%Slv_Addr
   MOV I2C_MCB+1,#%Count_1
   MOV I2C_MCB+2,#%Dest_Ptr_1
   MOV I2C_MCB+3,#%Sub_Addr
   MOV I2C_MCB+1,#%Count_2
   MOV I2C_MCB+2,#%Dest_Ptr_2
   ACALL _I2C_READ_REP_READ
   )

%* DEFINE (I2C_READ_REP_WRITE(Slv_Addr,Count_1,Dest_Ptr,Sub_Addr,Count_2,Source_Ptr))
   (
   MOV I2C_MCB  ,#%Slv_Addr
   MOV I2C_MCB+1,#%Count_1
   MOV I2C_MCB+2,#%Dest_Ptr
   MOV I2C_MCB+3,#%Sub_Addr
   MOV I2C_MCB+1,#%Count_2
   MOV I2C_MCB+2,#%Source_Ptr
   ACALL _I2C_READ_REP_WRITE
   )
```

I2C_PLM.H

```
/*=========================================================================*/
/*                                                                         */
/*      INCLUDE_FILE:   I2C_PLM.H                                          */
/*      PACKAGE:        I2C                                                */
/*                                                                         */
/*=========================================================================*/


/*=========================================================================*/
/*      G L O B A L   F U N C T I O N   P R O T O T Y P E S                */
/*=========================================================================*/
I2C_INIT: PROCEDURE(Own_Slv_Addr,Slv_Buf_Addr,Retry) BIT EXTERNAL;
    DECLARE(Own_Slv_Addr,Slv_Buf_Addr,Retry) BYTE MAIN;
END I2C_INIT;

I2C_TEST_DEVICE: PROCEDURE(Slv_Addr) BIT EXTERNAL;
    DECLARE(Slv_Addr) BYTE MAIN;
END I2C_TEST_Device

I2C_WRITE: PORCEDURE(Slv_Addr,Count,Source_Ptr) BIT EXTERNAL;
    DELCARE(Slv_Addr,Count,Source_Ptr) BYTE MAIN;
END I2C_WRITE;

I2C_WRITE_SUB; PROCEDURE(Slv_Addr,Count,Source_Ptr,Sub_Addr) BIT EXTERNAL;
    DECLARE(Slv_Addr,Count,Source_Ptr,Sub_Addr) BYTE MAIN;
END I2C_WRITE_SUB;

I2C_WRITE_SUB_SWINC: PROCEDURE(Slv_Addr,Count,Source_Ptr,Sub_Addr) BIT EXTERNAL;
    DECLARE(Slv_Addr,Count,Source_Ptr,Sub_Addr) BYTE MAIN;
END I2C_WRITE_SUB_SWINC;

I2C_WRITE_MEMORY: PROCEDURE(Slv_Addr,Count,Source_Ptr,Sub_Addr) BIT EXTERNAL;
    DECLARE(Slv_Addr,Count,Source_Ptr,Sub_Addr) BYTE MAIN;
END I2C_WRITE_MEMORY;

I2C_WRITE_SUB_WRITE: PROCEDURE(Slv_Addr,Count_1,Source_Ptr_1,Sub_Addr,Count_2,Source_Ptr_2) BIT EXTERNAL;
    DECLARE(Slv_Addr,Count-1,Source_Ptr_1,Sub_Addr,Count_2,Source_Ptr_2) BYTE MAIN;
END I2C_WRITE_SUB_WRITE;

I2C_WRITE_SUB_READ: PROCEDURE(Slv_Addr,Count_1,Source_Ptr,Sub_Addr,Count,Dest_Ptr) BIT EXTERNAL;
    DECLARE(Slv_Addr,Count_1,Source_Ptr,Sub_Addr,Count,Dest_Ptr) BYTE MAIN;
END I2C_WRITE_SUB_READ;

I2C_WRITE_COM_WRITE: PROCEDURE(Slv_Addr,Count_1,Source_Ptr_1,Count_2,Source_Ptr_2) BIT EXTERNAL;
    DECLARE(Slv_Addr,Count_1,Source_Ptr_1,Count_2,Source_Ptr_2) BYTE MAIN;
END I2C_WRITE_COM_WRITE;

I2C_WRITE_REP_WRITE: PROCEDURE(Slv_Addr,Count_1,Source_Ptr_1,Sub_Addr,Count_2,Source_Ptr_2) BIT EXTERNAL;
    DECLARE(Slv_Addr,Count_1,Source_Ptr_1,Sub_Addr,Count_2,Source_Ptr_2) BYTE MAIN;
END I2C_WRITE_REP_WRITE;

I2C_WRITE_REP_READ: PROCEDURE(Slv_Addr,Count_1,Source_Ptr,Sub_Addr,Count_2,Dest_Ptr) BIT EXTERNAL;
    DECLARE(Slv_Addr,Count_1,Source_Ptr,Sub_Addr,Count_2,Dest_Ptr) BYTE MAIN;
END I2C_WRITE_REP_READ;

I2C_READ: PROCEDURE(Slv_Addr,Count,Dest_Ptr) BIT EXTERNAL;
    DECLARE(Slv_Addr,Count,Dest_Ptr) BYTE MAIN;
END I2C_READ;

I2C_READ_STATUS: PROCEDURE(Slv_Addr,Dest_Ptr) BIT EXTERNAL;
    DECLARE(Slv_Addr,Dest_Ptr) BYTE MAIN;
END I2C_READ_STATUS;

I2C_READ_SUB: PROCEDURE(Slv_Addr,Count,Dest_Ptr,Sub_Addr) BIT EXTERNAL;
    DECLARE(Slv_Addr,Count,Dest_Ptr,Sub_Addr) BYTE MAIN;
END I2C_READ_SUB;

I2C_READ_REP_READ: PROCEDURE(Slv_Addr,Count_1,Dest_Ptr_1,Sub_Addr,Count_2,Dest_Ptr_2) BIT EXTERNAL;
    DECLARE(Slv_Addr,Count_1,Dest_Ptr_1,Sub_Addr,Count_2,Dest_Ptr_1) BYTE MAIN;
END I2C_READ_REP_READ;

I2C_READ_REP_WRITE: PROCEDURE(Slv_Addr,Count_1,Dest_Ptr_1,Sub_Addr,Count_2,Source_Ptr) BIT EXTERNAL;
    DECLARE(Slv_Addr,Count_1,Dest_Ptr_1,Sub_Addr,Cout_2,Source_Ptr) BYTE MAIN;
END I2C_READ_REP_WRITE;
```

I2C_C.H

```
/*===========================================================================*/
/*                                                                           */
/*      INCLUDE_FILE:   I2C_C.H                                              */
/*      PACKAGE:        I2C                                                  */
/*                                                                           */
/*===========================================================================*/


/*===========================================================================*/
/*      G L O B A L   F U N C T I O N   P R O T O T Y P E S                  */
/*===========================================================================*/

bit I2C_INIT(char Won_Slv_Addr,char *Slv_Buf_Ptr,char Retry);
bit I2C_TEST_Device(char Slv_Addr);
bit I2C_WRITE(char Slv_Addr,char Count,shar *Source_Ptr);
bit I2C_WRITE_SUB(char Slv_Addr,char Count,char *Source_Ptr,char Sub_Addr);
bit I2C_WRITE_SUB_SWINC(char Slv_Addr,char Count,char *Source_Ptr,char Sub_Addr);
bit I2C_WRITE_MEMORY(char Slv_Addr,char Count,char *Source_Ptr,char Sub_Addr);
bit I2C_WRITE_SUB_WRITE(char Slv_Addr,char Count_1,char *Source_Ptr_1,char Sub_Addr,char Count_2,char *Source_Ptr_2);
bit I2C_WRITE_SUB_READ(char Slv_Addr,char Count_1,char *Source_Ptr,char Sub_Addr,char Count,char *Dest_Ptr);
bit I2C_WRITE_COM_WRITE(char Slv_Addr,char Count_1,char *Source_Ptr_1,char Count_2,char *Source_Ptr_2);
bit I2C_WRITE_REP_WRITE(char Slv_Addr,char Count_1,char *Source_Ptr_1,char Sub_Addr,char Count_2,char *Source_Ptr_2);
bit I2C_WRITE_REP_READ(char Slv_Addr,char Count_1,char *Source_Ptr,char Sub_Addr,char Count_2,char *Dest_Ptr);
bit I2C_READ(char Slv_Addr,char Count,char *Dest_Ptr);
bit I2C_READ_STATUS(char Slv_Addr,char *Dest_Ptr);
bit I2C_READ_SUB(char Slv_Addr,char Count,char *Dest_Ptr,char Sub_Addr);
bit I2C_READ_REP_READ(char Slv_Addr,char Count_1,char *Dest_Ptr_1,char Sub_Addr,char Count_2,char *Dest_Ptr_2);
bit I2C_READ_REP_WRITE(char Slv_Addr,char Count_1,char *Dest_Ptr_1,char Sub_Addr,char Count_2,char *Source_Ptr);
```

DEMO_ASM.ASM

```
$CASE
$TITLE(Assembly example program)
;*=========================================================*
;*                                                         *
;*         SOURCE FILE : DEMO_ASM.ASM                      *
;*         PACKAGE     : I2C                               *
;*                                                         *
;* Hours and minutes will be displayed on LCD display      *
;* Dot between hours and minutes will blink                *
;*                                                         *
;*=========================================================*
$DEBUG
$NOLIST


;*=========================================================*
;*    I N C L U D E S                                      *
;*=========================================================*
$INCLUDE(I2C_DATA.GLO)
$INCLUDE(I2C_MAC.DEF)
$LIST


;*=========================================================*
;*    G L O B A L   F U N C T I O N   D E F I N I T I O N S *
;*=========================================================*
    EXTRN CODE(_I2C_INIT)
        EXTRN CODE(_I2C_WRITE)
        EXTRN CODE(_I2C_READ_SUB)


;*=========================================================*
;*    L O C A L   D A T A   D E F I N I T I O N S          *
;*=========================================================*
RAMVAR      SEGMENT DATA    ;Segment for variables
STACK       SEGMENT DATA    ;Segment for variables
USER        SEGMENT CODE    ;Segment for application program


;*=========================================================*
;*    L O C A L   S Y M B O L   D E F I N I T I O N S      *
;*=========================================================*
CLOCK_ADR   EQU 0A2H         ;Address of PCF8583
CL_SUB_ADR  EQU 01H          ;Sub address for reading time
LCD_ADR     EQU 74H          ;Address of PCF8577


;*=========================================================*
;*    D A T A   S E G M E N T                              *
;*=========================================================*
        RSEG RAMVAR
TIME_BUFFER:    DS 4                ;Buffer for I2C strings
LCD_BUFFER:     DS 5

        RSEG STACK
STACK_DATA:     DS 10


;*=========================================================*
;*    C O D E   S E G M E N T                              *
;*=========================================================*
        CSEG AT 00
        AJMP APPL_MAIN

        RSEG USER

APPL_MAIN:
        MOV  SP,#STACK_DATA-1
        MOV DPTR,#LCD_TAB           ;Pointer to segment table
        MOV LCD_BUFFER,#00          ;Ctrl word for LCD driver
        %I2C_INIT(22h,TIME_BUFFER,0)  ;Init I2C interface
        CLR A                       ;Prepare buffer
        MOV TIME_BUFFER,A           ;for clock int.
        MOV TIME_BUFFER+1A
        %I2C_WRITE(CLOCK_ADR,2,TIME_BUFFER) ;Initialize clock

REPEAT: %I2C_READ_SUB(CLOCK_ADR,4,TIME_BUFFER,CL_SUB_ADR)
                                    ;Read time
```

```
;*-------------------------------------------------------------*
;*      Time has been read. Order:                             *
;*               hundreds of sec's, sec's, min's and hr's      *
;*-------------------------------------------------------------*
        MOV A,TIME_BUFFER+3             ;Mask of hour counter
        ANL A,#3Fh
        MOV TIME_BUFFER+3,A

        ACALL CONVERT                   ;Convert time data to
                                        ;LCD segment data

;*-------------------------------------------------------------*
;*      Check if dot has to be switched on                     *
;*-------------------------------------------------------------*
        ORL LCD_BUFFER+3,#01h


;*-------------------------------------------------------------*
;*      If lsb of seconds is '0', then switch on dp            *
;*-------------------------------------------------------------*
        MOV A,TIME_BUFFER+1             ;Get seconds
        RRC A
        JC PROCEED
        ORD LCD_BUFFER+1,#01            ;Switch on dp


;*-------------------------------------------------------------*
;*      Display new time                                       *
;*-------------------------------------------------------------*
PROCEED: %I2C_WRITE(LCD_ADR,5,LCD_BUFFER)
        SJMP REPEAT                     ;Read new time


;*-------------------------------------------------------------*
;*      CONVERT converts BCD data of time to segment data      *
;*-------------------------------------------------------------*
CONVERT:MOV R0,#LCD_BUFFER+1            ;R0 is pointer
        MOV A,TIME_BUFFER+3             ;Get hours
        SWAP A                          ;Swap nibbles
        ACALL LCD_DATA                  ;Convrt 10's of hours
        MOV A,TIME_BUFFER+3
        ACALL LCD_DATA                  ;Convert hours
        MOV A,TIME_BUFFER+2             ;Get minutes
        SWAP A
        ACALL LCD_DATA                  ;Convrt 10's of minut
        MOV A,TIME_BUFFER+2
        ACALL LCD_DATA                  ;Convert minutes
        RET


;*-------------------------------------------------------------*
;*      LCD_DATA gets data from segment table and              *
;*      stores it in LCD_BUFFER                                *
;*-------------------------------------------------------------*
LCD_DATA:
        ANL A,#0FH                       ;Mask off LS-nibble
        MOVC A,@A+DPTR                   ;Get segment data
        MOV @R0,A                        ;Save segment data
        RET


;*-------------------------------------------------------------*
;*      Conversion table for LCD                               *
;*-------------------------------------------------------------*
LCD_TAB:
        DB 0FCH,60H,0DAH                 ;'0','1','2'
        DB 0F2H,66H,0B6H                 ;'3','4','5'
        DB 3EH,0E0H,0FEH                 ;'6','7','8'
        DB 0E6H                          ;'9'
;
        END
```

DEMO_PLM.PLM

```
$OPTIMIZE(4)
$DEBUG
$CODE
/*=========================================================*/
/*                                                         */
/*         SOURCE FILE : DEMO_PLM.PLM                      */
/*         PACKAGE     : I2C                               */
/*                                                         */
/* Hours and minutes will be displayed on LCD display      */
/* Dot between hours and minutes will blink                */
/*                                                         */
/*=========================================================*/


Demo_plm: Do;
/*=========================================================*/
/*    I N C L U D E S                                      */
/*=========================================================*/
$NOLIST
$INCLUDE(I2C_PLM.H)
$LIST


/*=========================================================*/
/*    M A I N                                              */
/*=========================================================*/

    Clock: Do;
          /* Variable and constand declarations */

          Declare LCD_TAB(*) Byte Constant (0FCh, 60H, 0DAH,
                         0F2H,66H,0B6H,3EH,0E0H,0FEH,0E6H);
          Declare Time_Buffer(4) Byte Main;
          Declare LCD_Buffer(5) Byte Main;
          Declare Tab_Point Word Main;
          Declare (LCD_Point,Time_Poiont) Byte Main;
          Declare Segment Based LCD_Point Byte Main;
          Declare Time Based Time_Point Byte Mian;
          Declare Tab_Value Based Tab_Point Byte Constant;

          Declare Clock_Adr Literally '0A2h';
          Declare LCD_Adr Literally '74h';
          Declare Cl_Sub_Adr Literally '01h';

          Call I2C_INIT(22h,.Time_Buffer,0);
          LCD_Buffer(0)=0; /* LCD control word */
          Time_Buffer(0)=0;
          Time_Buffer(1)=0;
          Call I2C_WRITE(Clock_Adr,2,.Time_Buffer);
                                        /* Initialize clock */
          Do While LCD_Buffer(0)=0; /* Program loop */
             Call I2C_READ_SUB(Clock_Adr,4,.Time_Buffer,
                               Cl_Sub_Adr);     /* Get time */
              LCD_Point=.LCD_Buffer(1); /* Init pointers */
              Time_point=.Time_Buffer(3);
              Tab_Point=.LCD_Tab(0)+SHR(Time,4); /* 10-HR's */
              Segment=Tab_Value;
              LCD_Point=LCD_Point+1;
              Tab_Point=.LCD_Tab(0)+(Time AND 0FH); /* HR's */
              Segment=Tab_Value;
              Time_Point=Time_Point-1;
              LCD_Point=LCD_Point+1;
              Tab_Point=.LCD_Tab+SHR(Time,4);   /* 10-MIN's */
              Segment=(Tab_Value OR 01H); /* dp */
              LCD_Point=LCD_Point+1;
              Tab_Point=.LCD_Tab+(Time AND 0FH);   /* MIN's */
              Segment=Tab_Value;
              Time_Pont=.Time_Buffer(1); /* Check sec's for
                                                 blinking */
              LCD_Point=.LCD_Buffer+1;
              If (Time AND 01H)>0 then
                  Segment=(Segment OR 01H);
              Call I2C_WRITE(LCD_Adr,5,.LCD_Buffer);
                                        /* Display time */
          End;

    End Clock;

End Demo_plm;
```

DEMO_C.C

```
/*=========================================================*/
/*                                                         */
/*         SOURCE FILE : DEMO_C.C                          */
/*         PACKAGE     : I2C_DEMO                          */
/*                                                         */
/*=========================================================*/

/*MPP:::xxxxxx=============================================*/
/*                                                         */
/* PACKAGE NAME: I2C_DEMO                                  */
/* DESCRIPTION:                                            */
/*    This demo program reads the time from a PCF8583 clock*/
/*    IC, and displays it to an LCD display (both available*/
/*    at the I2C demoboard.                                */
/*                                                         */
/*    Hours and minutes will be displayed on LCD display   */
/*    Dot between hours and minutes will blink             */
/*                                                         */
/*EMP=======================================================*/


/*=========================================================*/
/*    I N C L U D E S                                      */
/*=========================================================*/
    #include ”I2C_C.H”


/*=========================================================*/
/*    L O C A L   S Y M B O L   D E C L A R A T I O N S    */
/*=========================================================*/
    #define Clock_Adr       0xA2
    #define LCD_Adr         0x74
    #define Cl_Sub_Adr      0x01


/*=========================================================*/
/*    L O C A L   D A T A   D E F I N I T I O N S          */
/*=========================================================*/
    rom  char       LCD_Tab[]={0xFC,0x60,0xDA,0xF2,0x66,
                               0xB6,0x3E,0xE0,0xFE,0xE6};


/*=========================================================*/
/*    M A I N                                              */
/*=========================================================*/
void main()
{
    rom  char              *Tab_Ptr;
    data char              Time_Buffer[4];
    data char              *Time_Ptr;
    data char              LCD_Buffer[5];
    data char              *LCD_Ptr;

    I2C_INIT90x22,&Time_Buffer,0);
    LCD_Buffer[0]=0; /* LCD control word*/
    Time_Buffer[0]=0;
    Time_Buffer[1]=0;
    I2C_WRITE(Clock_Adr,2,&Time_Buffer); /* Init clock*/

    while (1)      /* Program loop*/
    {
        I2C_READ_SUB(Clock_Adr,4,&Time_Buffer,Cl_Sub_Adr);
                                                  /* Get time*/
        LCD_Ptr = &LCD_Buffer[1];   /* Init pointers*/
        Time_Ptr = &Time_Buffer[3];
        Tab_Ptr = (LCD_Tab+(*Time_Ptr >> 4));     /*10–HR’s*/
        *(LCD_Ptr++) =*Tab_Ptr;
        Tab_Ptr = (LCD_Tab+(*(Time_Ptr––) & 0x0F)); /* HR’s*/
        *(LCD_Ptr++) =*Tab_Ptr;
        Tab_Ptr = (LCD_Tab+(*Time_Ptr >> 4)); /* 10–MIN’s*/
        *LCD_Ptr++) = (*Tab_Ptr | 0x01); /* dp*/
        Tab_Ptr = (LCD_Tab+(*Time_Ptr & 0x0F)); /* MIN’s*/
        *LCD_Ptr =*Tab_Ptr;
        Time_Ptr = &Time_Buffer[1]; /* Check sec’s blinking*/
        LCD_Ptr = &LCD_Buffer[1];
        if ((*Time_Ptr & 0x01)>0)
            *LCD_Ptr = (*LCD_Ptr | 0x01);
        I2C_WRITE(LCD_Adr,5,&LCD_Buffer); /* Display time*/
    }
}
```

I²C driver routines for 8XC751/2 microcontrollers EIE/AN91007

Definitions

Short-form specification — The data in a short-form specification is extracted from a full data sheet with the same type number and title. For detailed information see the relevant data sheet or data handbook.

Limiting values definition — Limiting values given are in accordance with the Absolute Maximum Rating System (IEC 134). Stress above one or more of the limiting values may cause permanent damage to the device. These are stress ratings only and operation of the device at these or at any other conditions above those given in the Characteristics sections of the specification is not implied. Exposure to limiting values for extended periods may affect device reliability.

Application information — Applications that are described herein for any of these products are for illustrative purposes only. Philips Semiconductors make no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Disclaimers

Life support — These products are not designed for use in life support appliances, devices or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips Semiconductors customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors for any damages resulting from such application.

Right to make changes — Philips Semiconductors reserves the right to make changes, without notice, in the products, including circuits, standard cells, and/or software, described or contained herein in order to improve design and/or performance. Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no license or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

Philips Semiconductors
811 East Arques Avenue
P.O. Box 3409
Sunnyvale, California 94088–3409
Telephone 800-234-7381

Date of release: 06-98

Document order number: 9397 750 04053

Let's make things better.

Philips Semiconductors

PHILIPS