APPLICATION NOTE

C routines for the PCx8584

AN95068

Philips
Semiconductors

PHILIPS

C routines for the PCx8584

Application Note
AN95068

Abstract

The PCx8584 is designed to serve as an interface between most standard microcontrollers / processors and the I²C bus. This report describes worked-out driver software (written in C) to program the PCx8584 hardware registers. Also, interface routines offering the user a quick start in writing a complete I²C system application are described.

APPLICATION NOTE

C routines for the PCx8584

AN95068

Author(s):

Paul Seerden
Product Concept & Application Laboratory Eindhoven,
The Netherlands

Keywords

Microcontroller
Driver routines
Application software
I^2C bus
(multi)Master-Slave

Date: 1st September, 1995

Summary

This application note demonstrates how to write a driver, in C, for the Philips PCx8584 Inter Integrated Circuit bus (I^2C) controller. Also, a set of application interface software routines is given, to quickly implement a complete I^2C multi-master system application.

A small example program of how to use the driver is listed.

The driver supports i.a. polled or interrupt driven message handling, slave message transfers and multi-master system applications. Furthermore, it is suitable for use in conjunction with real time operating systems, like OS-9 or pSOS+.

CONTENTS

1. Introduction

This report describes driver routines, in C, for the PCx8584 I²C-bus controller. These driver routines are the interface between application software and the I²C device(s). These devices conform to the serial bus interface protocol specification as described in the I²C reference manual.

The I²C bus consists of two wires carrying information between the devices connected to the bus. Each device has its own 7-bit address. It can act as a master or as a slave during a data transfer. A master is the device that initiates the data transfer and generates the clock signals needed for the transfer. At that time any addressed device is considered a slave. The I²C bus is a multi-master bus. This means that more than one device capable of controlling the bus can be connected to it.

This driver supports both master and slave message transfers, as well as polled- and interrupt-driven message handling.

The driver is completely written in C programming language. Both the software structure and the interface to the application are described separately. The driver program has been tested as thoroughly as time permitted; however, Philips cannot guarantee that this I²C driver is flawless in all applications.

This application note (with C source files) is available for downloading from the PHIBBS (Philips Bulletin Board System). It is packed in the self extracting PC DOS file: PCx8584.EXE. The system is open to all callers, operates 24 hours a day and can be accessed with modems up to 28800 bps.

The BBS can be reached via telephone number: +31 40 2721102.

Used references:

- The I²C-bus specification	9398 358 10011
- The I²C-bus and how to use it	9398 393 40011
- Application report PCF8584 I2C-bus controller MAR 93	
- Specification I²C driver (J. Reitsma)	
- P90CL301 I²C driver routines	AN94078

Used development- and test tools:

- Microtec MCC68k C compiler (version 4.2I)	
- Philips Microcore 2 (SCC68070) evaluation board	OM4160/2
- Philips I²C-bus evaluation board	OM1016
- Philips Logic Analyzer PM3580/PM3585 with I²C-bus support package PF8681	

2. Functional description

2.1. The I²C bus format

An I²C transfer is initiated with the generation of a start condition. This condition will set the bus busy. After that a message is transferred that consists of an address and a number of data bytes. This I²C message may be followed either by a stop condition or a repeated start condition. A stop condition will release the bus mastership. A repeated start offers the possibility to send /receive more than one message to/from the same or different devices, while retaining bus mastership. Stop and (repeated) start conditions can only be generated in master mode.

Data and addresses are transferred in eight bit bytes, starting with the most significant bit. During the 9th clock pulse, following the data byte, the receiver must send an acknowledge bit to the transmitter. The clock speed is normally 100 KHz. Clock pulses may be stretched (for timing causes) by the slave.

A start condition is always followed by a 7-bits slave address and a R/W direction bit.

In a multi-master system, arbitration is done automatically by the I²C hardware. Arbitration can carry on through many bits, it can even continue into the comparison of data bytes. If arbitration is lost, a master mode error is generated and the driver switches to slave mode. After the slave transfer is ready, a switch back to master mode (retry) can be done.

General format and explanation of an I²C message:

S	SLV_W	**A**	SUB	**A**	**S**	SLV_R	**A**	*D1*	**A**	*D2*	**A**	. . . .	**A**	*Dn*	**N**	**P**

S	: (re)Start condition.
A	: Acknowledge on last byte.
N	: No Acknowledge on last byte.
P	: Stop condition.
SLV_W	: Slave address and Write bit.
SLV_R	: Slave address and Read bit.
SUB	: Sub-address.
D1 ... Dn	: Block of data bytes.

also:

D1.1 ... D1.m	: First block of data bytes.
Dn.1 ... Dn.m	: n_{th} block of data bytes.

2.2. Input definition

Inputs (application's view) to the driver are:

- The number of messages to exchange (transfer).
- The slave address of the I²C device for each message.
- The data direction (read/write) for all messages.
- The number of bytes in each message.
- In case of a write messages: the data bytes to be written.

2.3. Output definition

Outputs (application's view) from the driver are:

- Status information (success or error code).
- The number of messages actually transferred (not equal to the requested number of messages in case of an error).
- For each read message: The data bytes read.

2.4. Performance

The speed of the I²C-bus is controlled by the clock register S2 of the PCx8584. This register provides a prescaler that can be programmed to select one of five different clock rates, externally connected to pin 1 of the PCx8584. Furthermore, it provides a selection of four different I²C-bus SCL frequencies, ranging up to 90 KHz. The value for register S2 is passed as a parameter during initialization of the driver. To select the correct initialization values, refer to the Data sheet or the Application report of the PCx8584.

2.5. Using interrupts

Normally (default) the driver operates in polling mode. If a transfer is applied for, the driver interface function will not return until the transfer has ended.

To let the driver operate in interrupt driven mode, the *I2C_InstallInterrupt* function must be called after initialization. The PCx8584 is able to generate an interrupt vector during an interrupt acknowledge cycle. This interrupt vector is programmable (register S3) and is passed as parameter to the *I2C_InstallInterrupt* function. The interrupt vector is used by the microprocessor (or controller) to point to (using a vector table) the interrupt routine, called *I2C_Interrupt*.

If a transfer is started in interrupt driven mode, the driver interface function returns immediately.

At the end of the transfer (polled or interrupt driven), together with the generate stop condition, the driver calls a function, passing the transfer status, that was given by the application at the time the transfer was applied for. It's up to the user to write this function and to determine the actions that have to be done. (see example I2CINTFC.C).

2.6. Using an operating system

If you want to use this driver together with a multi-tasking (real time) operating system (like pSOS+), you only have to write or adjust the example interface file I2CINTFC.C (I2CDRIVR.C remains unchanged).

In the interface software (I2CINTFC.C) where the driver program is called, operating system calls have to be added. Examples of these calls are 'wait for/set semaphore' or 'send/receive mail', program time-outs, etc. This way other tasks in your system will not be blocked.

2.7. Error handling

A transfer 'status' is passed every time the 'transfer ready' function is called by the driver. It's up to the user to handle time outs, retries or all kind of other possible errors. Simple examples of these (no operating system, and no harware timers) are shown in the file I2CINTFC.C.

2.8. Interface mode control

Selection of either an Intel or Motorola bus interface is achieved by detection of the first WR - CS signal sequence (see data sheet). This driver assumes that previously the right interface is selected (after power-up).

3. External (application) interface

This section specifies the external interface of the driver towards the application. The C-coded external interface definitions are in the include file I2CEXPRT.H.

The application's view on the I²C bus is quite simple: The application can send messages to an I²C device. Also, the application must be able to exchange a group of messages, optionally addressed to different devices, without losing bus mastership. Retaining the bus is needed to guarantee atomic operations.

3.1. External data interface

All parameters affected by an I²C master transfer are logically grouped within two data structures. The user fills these structures and then calls the interface function to perform a transfer. The data structures used are listed below.

```
typedef struct
{
        BYTE            nrMessages;     /* total number of messages                              */
        I2C_MESSAGE     **p_message;    /* ptr to array of ptrs to message parameter blocks      */
} I2C_TRANSFER;
```

The structure I2C_TRANSFER contains the common parameters for an I²C transfer. The driver keeps a local copy of these parameters and leaves the contents of the structure unchanged. So, in many applications the structure only needs to be filled once.

After finishing the actual transfer, a 'transfer ready' function is called. The driver status and the number of messages done, are passed to this function.

The structure contains a pointer (p_message) to an array with pointers to the structure I2C_MESSAGE, shown below.

```
typedef struct
{
        BYTE    address;        /* The I2C slave device address          */
        BYTE    nrBytes;        /* number of bytes to read or write      */
        BYTE    *buf;           /* pointer to data array                 */
} I2C_MESSAGE;
```

The direction of the transfer (read or write) is determined by the lowest bit of the slave address;

write = 0 and read = 1. This bit must be (re)set by the application.

The array ***buf*** must contain data supplied by the application in case of a write transfer. The user should notice that checking to ensure that the buffer pointed to by ***buf*** is at least nrBytes in length, cannot be done by the driver.

In case of a read transfer, the array is filled by the driver. If you want to use ***buf*** as a string, a terminating NULL should be added at the end. It is the users responsibility to ensure that the buffer pointed to by ***buf*** is large enough to receive ***nrBytes*** bytes.

3.2. External function interfaces

This section gives a description of each callable interface function in the I²C driver module.

First the initialization functions are discussed. These functions directly program the I²C interface hardware and are part of the low level driver software. They must be called only once after 'reset', but before any transfer function is executed. The driver contains the following three initialization functions:

- *I2C_InitializeMaster* (in file I2CMASTR.C)
- *I2C_InitializeSlave* (in file I2CSLAVE.C)
- *I2C_InstallInterrupt* (in file I2CDRIVR.C)

void I2C_InitializeMaster(BYTE speed)

Initialize the I²C-bus **master** driver part. Hardware I²C registers of the PCx8584 interface will be programmed. Used constants (parameters) are defined in the file I2CDRIVR.H. Must be called once after RESET, before any other interface function is called.

BYTE **speed** Contents for clock register S2 (bit rate of I²C-bus).

*void I2C_InitializeSlave(BYTE ownAddress, BYTE *buf, BYTE size, BYTE speed)*

Initialize the I²C-bus **slave** driver part. Hardware I²C registers of the PCx8584 interface will be programmed with the designated parameters. Must be called once after RESET.

BYTE **ownAddress**	Micro-controller's own slave-address.
BYTE ***buf**	Pointer to buffer, to transmit data from, or receive data in.
BYTE **size**	Size of buffer to transmit data from, or receive data in.
BYTE **speed**	Contents for clock register S2 (bit rate of I²C-bus).

void I2C_InstallInterrupt(BYTE vector)

Install the I²C interrupt, using the specified priority. Must be called once after one of the initialization functions is called.

BYTE **vector**	Contents for vector register S3 (Interrupt vector).

Next two functions to 'perform transfers' will be discussed.

- *I2C_ProcessSlave* (in file I2CSLAVE.C)
- *I2C_Transfer* (in file I2CMASTR.C)

void I2C_ProcessSlave(void)

This function can be used by the application to handle slave transfers. It is just an example and should be customized by the user. Depending of the status of the slave it takes action. Possible slave states are:

SLAVE_IDLE	Slave is idle.
SLAVE_TRX	Slave transmitter mode.
SLAVE_RCV	Slave receiver mode.
SLAVE_LAST	Slave receiving last byte (send no ack).
SLAVE_READY	Slave transfer ready.

*void I2C_Transfer(I2C_TRANSFER *p, void (*proc)(BYTE status, BYTE msgsDone))*

Start a synchronous I²C transfer. When the transfer is completed, with or without an error, call the function ***proc***, passing the transfer status and the number of messages successfully transferred.

I2C_TRANSFER ***p**	A pointer to the structure describing the I²C messages to be transferred.
void (***proc**(status, msgsDone))	A pointer to the function to be called when the transfer is completed.

BYTE **msgsDone**	Number of message successfully transferred.	
BYTE **status**	one of:	
	I2C_OK	Transfer ended No Errors
	I2C_BUSY	I²C busy, so wait
	I2C_ERR	General error
	I2C_NO_DATA	err: No data message block
	I2C_NACK_ON_DATA	err: No ack on data in block
	I2C_NACK_ON_ADDRESS	err: No ack of slave
	I2C_DEVICE_NOT_PRESENT	err: Device not present
	I2C_ARBITRATION_LOST	err: Arbitration lost
	I2C_TIME_OUT	err: Time out occurred
	I2C_SLAVE_ERROR	Slave mode error
	I2C_INIT_ERROR	err: Initialization (not done)

3.3. Interface layer example

The module I2CINTFC.C gives an example of how to implement a few basic transfer functions (see also previous PCALE I²C driver application notes). These functions allow you to communicate with most of the available I²C devices and serve as a *layer* between your application and the driver software. This *layered approach* allows support for new devices (micro-controllers) without re-writing the high-level (device-independent) code. The given examples are:

```
void I2C_Write(I2C_MESSAGE *msg)
void I2C_WriteRepWrite(I2C_MESSAGE *msg1, I2C_MESSAGE *msg2)
void I2C_WriteRepRead(I2C_MESSAGE *msg1, I2C_MESSAGE *msg2)
void I2C_Read(I2C_MESSAGE *msg)
void I2C_ReadRepRead(I2C_MESSAGE *msg1, I2C_MESSAGE *msg2)
void I2C_ReadRepWrite(I2C_MESSAGE *msg1, I2C_MESSAGE *msg2)
```

Furthermore, the module I2CINTFC.C contains the functions *StartTransfer*, in which the actual call to the driver program is done, and the function *I2cReady*, which is called by the driver after the completion of a transfer. The flag **drvStatus** is used to test/check the state of a transfer.

In the *StartTransfer* function a software time-out loop is programmed. Inside this time-out loop the *MainStateHandler* is called if the driver is in polling mode and the status register PIN flag is set.

If a transfer has failed (error or time-out) the *StartTransfer* function prints an error message (using standard I/O redirection, *printf()* function) and it does a retry of the transfer. However, if the maximum number of retries is reached an exception interrupt (Trap #15) is generated to give a fatal error message.

4. Master operation

The PCx8584 logic provides a serial interface that meets the I²C bus specification and supports all master transfer modes from and to the bus.

A microcontroller / processor interfaces to the PCx8584 via five hardware registers: S0 (data read / write register), S0' (own address register), S1 (control / status register), S2 (clock register) and S3 (interrupt vector register).

After completing the transmission or reception of each byte (address or data), the PIN flag in the control / status register is reset to 0. In interrupt driven mode, an interrupt is sent to the micro and the interrupt service handler will be called. In polling mode this is done by software. In master mode this handler can be in one of the following states:

ST_IDLE	The state handler does not expect any interrupt.
ST_AWAIT_ACK	The driver has sent the slave address and waits for an acknowledge.
ST_RECEIVING	The handler is receiving bytes, and there is still more than one expected.
ST_RECV_LAST	The handler is waiting for the last byte to receive.
ST_SENDING	The handler is busy sending bytes to a device.

Figure 1 shows the state diagram of the master state handler. A state transition will occur on initiation of a transfer by the application and on each I²C (PIN goes low) interrupt. The transitions are:

ST_IDLE → ST_SENDING	A transfer is initiated. Send the slave address for the first write message.
ST_IDLE → ST_AWAIT_ACK	A transfer is initiated. A message is to be received from a slave device. The Micro transmits the slave address.
ST_SENDING → ST_SENDING	At least one byte to sent. Send the next byte. Or no more bytes to sent, send repeated start and slave addres of next message to write.
ST_SENDING → ST_IDLE	No more byte to sent, no more messages.
ST_SENDING → ST_AWAIT_ACK	No more bytes to sent, send repeated start and slave address of next message is to be received.
ST_AWAIT_ACK → ST_RECEIVING	More than 1 byte is to be received. Wait for and acknowledge next byte.
ST_AWAIT_ACK → ST_RECV_LAST	Only one byte to receive, send no acknowledge on last byte.

ST_RECEIVING → ST_RECEIVING	More than one byte to receive. Read received byte.
ST_RECEIVING → ST_RECV_LAST	Only one byte left to receive, send no acknowledge on it.
ST_RECV_LAST → ST_IDLE	Last byte read, send stop. No more messages.
ST_RECV_LAST → ST_SENDING	Last byte read, send repeated start and slave address of next (write) message.
ST_RECV_LAST → ST_AWAIT_ACK	Last byte read. send repeated start and slave address of next (read) message.

The procedure *MainStateHandler* (in I2CDRIVR.C) checks the statusflag 'master' and the loast arbitration bit, after that either the function *HandleMasterState* (in I2CMASTR.C) or the function *HandleSlaveState* (in I2CSLAVE.C) is called. Calling of these functions is done via two (initialized) pointers, masterProc and slaveProc.

If a master transfer is completed a function (*readyProc*) in the application (or interface) is called.

If a slave transfer is completed the slave status SLAVE_READY is set (see also section 3.2).

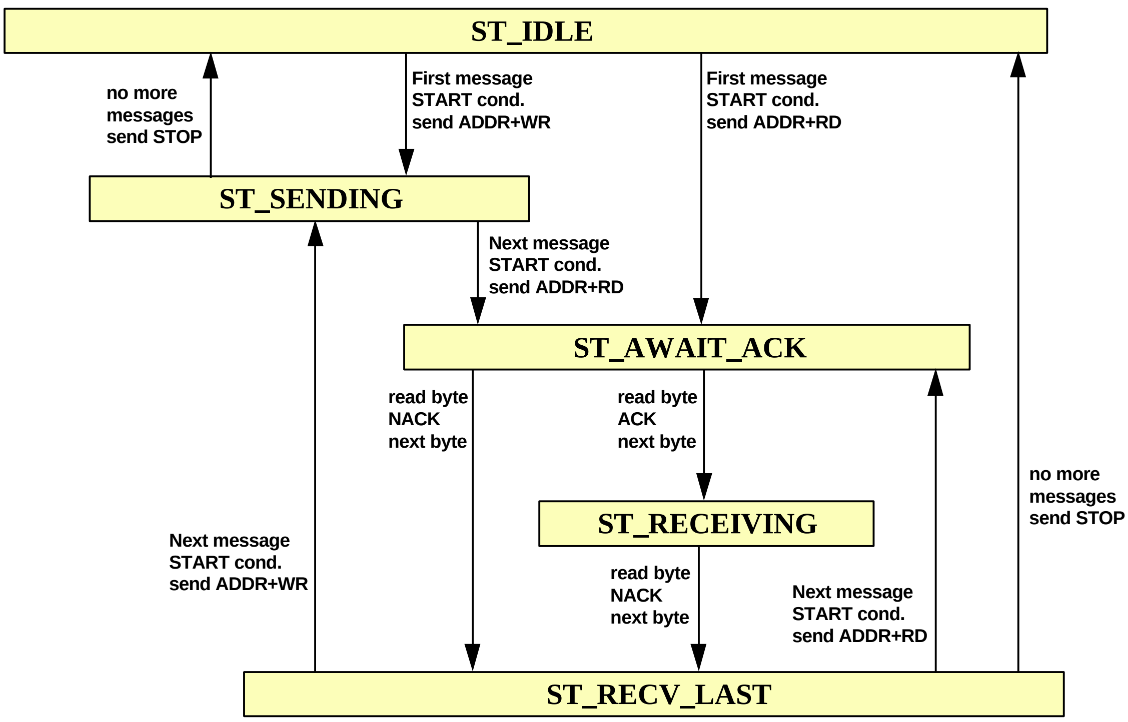

Figure 1. State transition diagram of the master state handler.

5. Slave operation

The slave-mode protocol is very specific to the system design, and therefore, very difficult to make generic.

In this report basic slave-receive and slave-transmit routines are given, but they only should be considered as examples. To activate the slave mode driver, call the function *I2C_InitializeSlave* (see also section 3.2). All slave routines are placed together in the module I2CSLAVE.C, this module is listed in appendix III.

There are two ways for the driver to enter the slave functions:

- Through a normal I²C interrupt (or polling the slave) when the driver is idle (in slave receiver mode) and the interface recognizes its own slave address, or a general call address.

- Through master mode, during transmission of a slave-address in master mode arbitration is lost to another master. The driver must then switch to slave-receiver mode to check if this other master wants to address him.

The slave routines as given, make use of a single data buffer. This buffer (pointer and size) is initialized during the *I2C_InitializeSlave* function.

When addressed as slave transmitter, bytes from the data buffer are transmitted until a NACK (No Acknowledge) or a stop condition is received.

When addressed as slave receiver, bytes from the I²C-bus are received into the data buffer until it is full (*size* is reached). The transfer is stopped by the driver by giving no acknowledge on the last data byte.

After a slave transfer the application must service the slave (i.e. process received data or put new data in the buffer). This is very application dependent, therefore the example function *I2C_ProcessSlave* must be customized by the user.

6. Modelling hierarchy

This I²C driver consists of 3 parts:

- Driver software; Initialization, Master functions, Slave functions.
- Interface functions; External application interface to the driver.
- An application example; Tested on the Microcore 2 (is a SCC68070 evaluation tool).

The driver package contains the following files:

- **I2CDRIVR.C**	The general driver needed for both master and slave features, containing the interrupt installation and handler. Always link this module to your application.
- **I2CMASTR.C**	The actual driver for master transfers, containing initialization and master state handling. Only needed if your PCx8584 acts as a bus master.
- **I2CSLAVE.C**	The actual driver functions needed for the micro + PCx8584 to act as a slave on the bus, containing initialization and state handling.
- **I2CDRIVR.H**	This module (include file) contains definitions of local data types and constants, and is used only by the driver package.
- **I2CINTFC.C**	This module contains **example** application interface functions to perform a master transfer. In this module some often used message protocols are implemented. Furthermore, it shows examples of error handling, like: time-outs (software loops), retries and error messages. The user must adapt these functions to his own system needs and environment.
- **I2CEXPRT.H**	This module (include file) contains definitions of all 'global' constants, function prototypes, data types and structures needed by the user (application). Include this file in the user application source files.
- **REG8584.H**	This module (include file) contains address definitions of hardware registers of the PCx8584. The user should adapt these definitions to his own system environment (address map).
- **EXAMPLE.C**	This program uses the driver package to implement a simple application with the PCx8584 and an I²C demonstration board.

Appendices

APPENDIX I I2CINTFC.C

```
/***************************************************************************/
/*  Acronym           : I²C  Inter IC bus (for PCx8584)                    */
/*  Name of module    : I2CINTFC.C                                          */
/*  Creation date     : 1995-08-01                                          */
/*  Program language  : C                                                   */
/*  Name              : P.H. Seerden                                        */
/*                                                                          */
/*  Description       : External interface to the PCx8584 I²C driver      */
/*                        routines. This module contains the EXAMPLE       */
/*                        interface functions, used by the application to  */
/*                        do I²C master-mode transfers.                     */
/*                                                                          */
/*                                                                          */
/*          (C) Copyright 1995 Philips Semiconductors B.V.                 */
/*              Product Concept & Application Laboratory Eindhoven (PCALE)  */
/*              Eindhoven - The Netherlands                                 */
/*                                                                          */
/***************************************************************************/
/*                                                                          */
/*  History:                                                                */
/*                                                                          */
/*  95-08-01    P.H. Seerden    Initial version                            */
/*                                                                          */
/*                                                                          */
/***************************************************************************/


#include "i2cexprt.h"
#include "i2cdrivr.h"
#include "reg8584.h"


static BYTE drvStatus;                    /* Status returned by driver      */

static I2C_MESSAGE   *p_iicMsg[2];  /* pointer to an array of (2) I2C mess */
static I2C_TRANSFER  iicTfr;



static void I2cReady(BYTE status, BYTE msgsDone)
/************************************************
 * Input(s)   : status       Status of the driver at completion time
 *              msgsDone     Number of messages completed by the driver
 * Output(s)  : None.
 * Returns    : None.
 * Description: Signal the completion of an I²C transfer. This function is
 *              passed (as parameter) to the driver and called by the
 *              drivers state handler (!).
 ***************************************************************************/
{
    drvStatus = status;
}
```

```
static void StartTransfer(void)
/******************************
 * Input(s)   : None.
 * Output(s)  : statusfield of I2C_TRANSFER contains the driver status:
 *                 I2C_OK          Transfer was successful.
 *                 I2C_TIME_OUT    Timeout occurred
 *                 Otherwise       Some error occurred.
 * Returns    : None.
 * Description: Start I²C transfer and wait (with timeout) until the
 *              driver has completed the transfer(s).
 ***************************************************************************/
{
  LONG timeOut;
  BYTE retries = 0;

    do
    {
        drvStatus = I2C_BUSY;
        I2C_Transfer(&iicTfr, I2cReady);

        timeOut = 0;
        while (drvStatus == I2C_BUSY)
        {
            if (++timeOut > 60000)
                drvStatus = I2C_TIME_OUT;
            if (intMask == 0)                  /* 0 -> polling             */
            {
                if (!(CR_8584 & PIN_MASK))     /* wait until PIN bit is 0  */
                    MainStateHandler();
            }
        }

        if (retries == 6)
        {
            printf("retry counter expired\n");   /* fatal error ! So, ..   */
            asm (" trap  #15 ");              /* escape to debug monitor    */
        }
        else
            retries++;

        switch (drvStatus)
        {
          case I2C_OK                 : break;
          case I2C_NO_DATA            : printf("buffer empty\n");       break;
          case I2C_NACK_ON_DATA       : printf("no ack on data\n");     break;
          case I2C_NACK_ON_ADDRESS    : printf("no ack on address\n");  break;
          case I2C_DEVICE_NOT_PRESENT : printf("device not present\n"); break;
          case I2C_ARBITRATION_LOST   : printf("arbitration lost\n");   break;
          case I2C_TIME_OUT           : printf("time-out\n");           break;
          default                     : printf("unknown status\n");     break;
        }
    } while (drvStatus != I2C_OK);
}
```

```
void I2C_Write(I2C_MESSAGE *msg)
/*******************************
 * Input(s)   : msg     I²C message
 * Returns    : None.
 * Description: Write a message to a slave device.
 * PROTOCOL   : <S><SlvA><W><A><D1><A> ... <Dnum><N><P>
 ****************************************************************************/
{
    iicTfr.nrMessages = 1;
    iicTfr.p_message  = p_iicMsg;
    p_iicMsg[0] = msg;

    StartTransfer();
}


void I2C_WriteRepWrite(I2C_MESSAGE *msg1, I2C_MESSAGE *msg2)
/**********************************************************
 * Input(s)   : msg1     first I²C message
 *               msg2     second I²C message
 * Returns    : None.
 * Description: Writes two messages to different slave devices separated
 *               by a repeated start condition.
 * PROTOCOL   : <S><Slv1A><W><A><D1><A>...<Dnum1><A>
 *                           <S><Slv2A><W><A><D1><A>...<Dnum2><A><P>
 ****************************************************************************/
{
    iicTfr.nrMessages = 2;
    iicTfr.p_message  = p_iicMsg;
    p_iicMsg[0] = msg1;
    p_iicMsg[1] = msg2;

    StartTransfer();
}


void I2C_WriteRepRead(I2C_MESSAGE *msg1, I2C_MESSAGE *msg2)
/*********************************************************
 * Input(s)   : msg1     first  I²C message
 *               msg2     second I²C message
 * Returns    : None.
 * Description: A message is sent and received to/from two different
 *               slave devices, separated by a repeat start condition.
 * PROTOCOL   : <S><Slv1A><W><A><D1><A>...<Dnum1><A>
 *                        <S><Slv2A><R><A><D1><A>...<Dnum2><N><P>
 ****************************************************************************/
{
    iicTfr.nrMessages = 2;
    iicTfr.p_message  = p_iicMsg;
    p_iicMsg[0] = msg1;
    p_iicMsg[1] = msg2;

    StartTransfer();
}


void I2C_Read(I2C_MESSAGE *msg)
/******************************
 * Input(s)   : msg     I²C message
 * Returns    : None.
 * Description: Read a message from a slave device.
 * PROTOCOL   : <S><SlvA><R><A><D1><A> ... <Dnum><N><P>
 ****************************************************************************/
{
    iicTfr.nrMessages = 1;
    iicTfr.p_message  = p_iicMsg;
    p_iicMsg[0] = msg;

    StartTransfer();
}
```

```
void I2C_ReadRepRead(I2C_MESSAGE *msg1, I2C_MESSAGE *msg2)
/**********************************************************
 * Input(s)   : msg1     first I²C message
 *              msg2     second I²C message
 * Returns    : None.
 * Description: Two messages are read from two different slave devices,
 *              separated by a repeated start condition.
 * PROTOCOL   : <S><Slv1A><R><A><D1><A>...<Dnum1><N>
 *                          <S><Slv2A><R><A><D1><A>...<Dnum2><N><P>
 ***************************************************************************/
{
    iicTfr.nrMessages = 2;
    iicTfr.p_message  = p_iicMsg;
    p_iicMsg[0] = msg1;
    p_iicMsg[1] = msg2;

    StartTransfer();
}


void I2C_ReadRepWrite(I2C_MESSAGE *msg1, I2C_MESSAGE *msg2)
/**********************************************************
 * Input(s)   : msg1     first I²C message
 *              msg2     second I²C message
 * Returns    : None.
 * Description: A block data is received from a slave device, and also
 *              a(nother) block data is send to another slave device
 *              both blocks are seperated by a repeated start.
 * PROTOCOL   : <S><Slv1A><R><A><D1><A>...<Dnum1><N>
 *                          <S><Slv2A><W><A><D1><A>...<Dnum2><A><P>
 ***************************************************************************/
{
    iicTfr.nrMessages = 2;
    iicTfr.p_message  = p_iicMsg;
    p_iicMsg[0] = msg1;
    p_iicMsg[1] = msg2;

    StartTransfer();
}
```

APPENDIX II I2CMASTR.C

```
/****************************************************************************/
/*  Acronym           : I2C  Inter IC bus (for PCx8584)                     */
/*  Name of module    : I2CMASTR.C                                          */
/*  Creation date     : 1995-08-01                                          */
/*  Program language  : C                                                   */
/*  Name              : P.H. Seerden                                        */
/*                                                                          */
/*                                                                          */
/*        (C) Copyright 1995 Philips Semiconductors B.V.                    */
/*            Product Concept & Application Laboratory Eindhoven (PCALE)    */
/*            Eindhoven - The Netherlands                                   */
/*                                                                          */
/****************************************************************************/
/*                                                                          */
/*  Description:                                                            */
/*                                                                          */
/*   Master driver part for the Philips PCx8584 I²C bus controller.        */
/*                                                                          */
/*   Everything between one Start and Stop condition is called a TRANSFER. */
/*   One transfer consists of one or more MESSAGEs.                        */
/*   To start a transfer call function "I2C_Transfer".                     */
/*                                                                          */
/*                                                                          */
/****************************************************************************/
/*                                                                          */
/*  History:                                                                */
/*                                                                          */
/*  95-08-01    P.H. Seerden    Initial version                             */
/*                                                                          */
/*                                                                          */
/****************************************************************************/


#include "i2cexprt.h"
#include "i2cdrivr.h"
#include "reg8584.h"


extern void (*masterProc)();          /* Handle Master Transfer action     */
extern void (*slaveProc)();           /* Handle Slave Transfer action      */

static I2C_TRANSFER *tfr;             /* Ptr to active transfer block      */
static I2C_MESSAGE  *msg;             /* ptr to active message block       */

static void (*readyProc)(BYTE,BYTE);  /* proc. to call if transfer ended   */
static BYTE mssgCount;                /* Number of messages sent           */
static BYTE dataCount;                /* nr of bytes of current message    */
static BYTE state;                    /* state of the I²C driver           */


static void GenerateStop(BYTE status)
/***********************************
 * Input(s)    : status       status of the driver.
 * Output(s)   : None.
 * Returns     : None.
 * Description : Generate a stop condition.
 ***************************************************************************/
{
    CR_8584 = PIN_MASK | ESO_MASK | intMask | STO_MASK | ACK_MASK;
    master  = FALSE;
    state   = ST_IDLE;

    readyProc(status, mssgCount);          /* Signal driver is finished    */
}
```

```
static void HandleMasterState(void)
/**********************************
 * Input(s)    : None.
 * Returns     : None.
 * Description : Master mode state handler for I2C bus.
 ***********************************************************************/
{
    if (CS_8584 & LAB_MASK)
    {                                       /* arbitration was lost          */
        slaveProc();                       /* check if addressed as slave  */
        GenerateStop(I2C_ARBITRATION_LOST);        /* leave the bus         */
        return;
    }
    switch (state)
    {
     case ST_SENDING :
        if (CS_8584 & LRB_MASK)
            GenerateStop(I2C_NACK_ON_DATA);
        else
            if (dataCount < msg->nrBytes)
                DR_8584 = msg->buf[dataCount++];       /* sent next byte   */
            else
                if (mssgCount < tfr->nrMessages)
                {
                    dataCount = 0;
                    msg = tfr->p_message[mssgCount++];
                    state = (msg->address & 1) ? ST_AWAIT_ACK : ST_SENDING;
                    CS_8584 = ESO_MASK | STA_MASK | ACK_MASK | intMask;
                    DR_8584 = msg->address;
                }
                else
                    GenerateStop(I2C_OK);              /* transfer ready   */
        break;
     case ST_AWAIT_ACK :
        if (CS_8584 & LRB_MASK)
            GenerateStop(I2C_NACK_ON_ADDRESS);
        else
        {
          BYTE dummy;
            if (msg->nrBytes == 1)
            {
                CS_8584 = ESO_MASK | intMask;          /* clear ACK        */
                state = ST_RECV_LAST;
            }
            else
                state = ST_RECEIVING;
            dummy = DR_8584;      /* generate clk pulses for first byte    */
        }
        break;
     case ST_RECEIVING :
        if (dataCount + 2 == msg->nrBytes)
        {
            CS_8584 = ESO_MASK | intMask;              /* clear ACK        */
            state = ST_RECV_LAST;
        }
        msg->buf[dataCount++] = DR_8584;
        break;
     case ST_RECV_LAST :
        if (mssgCount < tfr->nrMessages)
        {
            msg->buf[dataCount] = DR_8584;
            dataCount = 0;
            msg = tfr->p_message[mssgCount++];
            state = (msg->address & 1) ? ST_AWAIT_ACK : ST_SENDING;
            CS_8584 = ESO_MASK | STA_MASK | ACK_MASK | intMask;
            DR_8584 = msg->address;
        }
        else
        {
            GenerateStop(I2C_OK);                  /* transfer ready       */
            msg->buf[dataCount] = DR_8584;
        }
        break;
     default :                              /* impossible                   */
        GenerateStop(I2C_ERR);             /* just to be sure              */
        break;
    }
}
```

```
void I2C_Transfer(I2C_TRANSFER *p, void (*proc)(BYTE, BYTE))
/***********************************************************
 * Input(s)   : p            address of I²C transfer parameter block.
 *              proc         procedure to call when transfer completed,
 *                           with the driver status passed as parameter.
 * Output(s)  : None.
 * Returns    : None.
 * Description: Start an I²C transfer, containing 1 or more messages. The
 *              application must leave the transfer parameter block
 *              untouched until the ready procedure is called.
 ***************************************************************************/
{
    tfr = p;
    readyProc = proc;
    mssgCount = 0;
    dataCount = 0;
    master    = TRUE;
    msg = tfr->p_message[mssgCount++];

    state   = (msg->address & 1) ? ST_AWAIT_ACK : ST_SENDING;
    CS_8584 = ESO_MASK | STA_MASK | ACK_MASK | intMask;  /* generate start */
    DR_8584 = msg->address;
}


void I2C_InitializeMaster(BYTE speed)
/*************************************
 * Input(s)   : speed      clock register value for bus speed.
 * Output(s)  : None.
 * Returns    : None.
 * Description: Initialize the PCF8584 as I2C bus master.
 ***************************************************************************/
{
    state     = ST_IDLE;
    readyProc = NULL;

    masterProc = HandleMasterState;       /* Set pointer to correct proc.  */

    AR_8584 = 0x26;                        /* dummy own slave address       */

    CR_8584 = 0x20;                        /* write clock register          */
    CL_8584 = speed;

    CR_8584 = ESO_MASK;                    /* enable serial interface       */

    intMask = 0;
    master  = FALSE;
}
```

APPENDIX III I2CSLAVE.C

```
/***************************************************************************/
/*  Acronym            : I2C  Inter IC bus (for PCF8584)                   */
/*  Name of module     : I2CSLAVE.C                                        */
/*  Scope              : Application software                              */
/*  12nc               : xxxx xxx xxxx.x                                   */
/*  Creation date      : 1995-08-01                                        */
/*  Program language   : C                                                 */
/*  Name               : P.H. Seerden                                      */
/*                                                                         */
/*                                                                         */
/*        (C) Copyright 1995 Philips Semiconductors B.V.                   */
/*            Product Concept & Application Laboratory Eindhoven (PCALE)   */
/*            Eindhoven - The Netherlands                                  */
/*                                                                         */
/***************************************************************************/
/*                                                                         */
/*  Description:                                                           */
/*                                                                         */
/*     Part of the I²C driver that handles slave bus-transfers.            */
/*                                                                         */
/***************************************************************************/
/*                                                                         */
/*  History:                                                               */
/*                                                                         */
/*  95-08-01    P.H. Seerden    Initial version                            */
/*                                                                         */
/*                                                                         */
/***************************************************************************/


#include "i2cexprt.h"
#include "i2cdrivr.h"
#include "reg8584.h"


extern void  (*slaveProc)();     /* Handle Slave Transfer action           */


static BYTE  count;              /* bytes send/received of current message */
static BYTE  slaveStatus;        /* status of the slave                    */
static BYTE  size;               /* size of slave mode buffer              */
static BYTE  *slaveBuf;          /* ptr to rec/trm data into/from if slave */
```

```
void HandleSlaveState(void)
/**************************
 * Input(s)    : None.
 * Output(s)   : None.
 * Returns     : None.
 * Description: Procedure to handle actions if addressed as slave.
 ***************************************************************************/
{
    switch (slaveStatus)
    {
      case SLAVE_IDLE:
          if (CS_8584 & AAS_MASK)                 /* addressed as slave ?  */
          {
              count = 0;
              if (DR_8584 & 1)
              {
                  slaveStatus = SLAVE_TRX;           /* slave transmitter  */
                  DR_8584 = slaveBuf[count++];       /* sent first byte    */
              }
              else                                   /* slave receiver     */
                  if (size > 1)
                  {
                      slaveStatus = SLAVE_RCV;
                      CS_8584 = PIN_MASK | ESO_MASK | ACK_MASK | intMask;
                  }
                  else
                  {
                      slaveStatus = SLAVE_LAST;
                      CS_8584 = PIN_MASK | ESO_MASK | intMask;
                  }
          }
          break;
      case SLAVE_TRX:
          if (CS_8584 & LRB_MASK)
          {
              CS_8584 = PIN_MASK;                 /* no ack from master     */
              slaveStatus = SLAVE_READY;          /* last byte transmitted  */
          }
          else
              DR_8584 = slaveBuf[count++];        /* sent next byte         */
          break;
      case SLAVE_RCV:
          slaveBuf[count++] = DR_8584;
          if (count == size)
          {
              CS_8584 = ESO_MASK | intMask;                /* clear ACK     */
              slaveStatus = SLAVE_LAST;
          }
          else
              CS_8584 = ESO_MASK | ACK_MASK | intMask;    /* set ACK       */
          break;
      case SLAVE_LAST:
          slaveBuf[count] = DR_8584;
          CS_8584 = ESO_MASK | ACK_MASK | intMask;        /* set ACK       */
          slaveStatus = SLAVE_READY;                 /* last byte received  */
          break;
      default:
          CS_8584 = PIN_MASK;                        /* clear interrupt     */
          break;
    }
}
```

```
void I2C_InitializeSlave(BYTE slv, BYTE *buf, BYTE len, BYTE speed)
/*****************************************************************
 * Input(s)    : slv        Own slave address
 *               buf        Pointer to slave data buffer
 *               size       size of the slave data buffer
 *               speed      clock register value for bus speed
 * Output(s)   : None.
 * Returns     : None.
 * Description : Enable I²C (slave) bus and set the clock speed for I²C.
 **************************************************************************/
{
    slaveProc = HandleSlaveState;         /* Set pointer to correct proc.  */

    CR_8584 = 0;                           /* disable i2c interface         */
    AR_8584 = slv;                         /* write own slave address       */

    CR_8584 = 0x20;                        /* write clock register          */
    CL_8584 = speed;

    intMask  = 0;
    size     = len;
    slaveBuf = buf;
    slaveStatus = SLAVE_IDLE;

    CR_8584 = ESO_MASK;                    /* enable serial interface       */
}


void I2C_ProcessSlave(void)
/*************************
 * Input(s)   : None.
 * Output(s)  : None.
 * Returns    : None.
 * Description: Process the slave.
 *              This function must be called by the application to check
 *              the slave status. The USER should adapt this function to
 *              his personal needs (take the right action at a certain
 *              status).
 **************************************************************************/
{
    switch(slaveStatus)
    {
    case SLAVE_IDLE :
        /* do nothing or fill transmit buffer for transfer                 */
        break;
    case SLAVE_TRX  :
    case SLAVE_RCV  :
    case SLAVE_LAST :
        /* do nothing if interrupt driven, else poll PIN bit               */
        if (intMask == 0)                     /* 0 -> polling              */
        {
            if (!(CR_8584 & PIN_MASK))        /* wait until PIN bit is 0   */
                MainStateHandler();
        }
        break;
    case SLAVE_READY :
        /* read or fill buffer for next transfer, signal application       */
        slaveStatus = SLAVE_IDLE;
        break;
    }
}
```

APPENDIX IV I2CDRIVR.C

```
/*************************************************************************/
/*  Acronym           : I2C  Inter IC bus (for PCF8584)                  */
/*  Name of module    : I2CDRIVR.C                                       */
/*  Creation date     : 1995-08-01                                       */
/*  Program language  : C                                                */
/*  Name              : P.H. Seerden                                     */
/*                                                                       */
/*                                                                       */
/*        (C) Copyright 1995 Philips Semiconductors B.V.                 */
/*            Product Concept & Application Laboratory Eindhoven (PCALE) */
/*            Eindhoven - The Netherlands                                */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/*  Description:                                                         */
/*                                                                       */
/*     Main part of the I²C driver.                                      */
/*         Contains the interrupt handler and does calls to the master   */
/*         and/or slave driver part.                                     */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/*  History:                                                             */
/*                                                                       */
/*  95-08-01    P.H. Seerden    Initial version                          */
/*                                                                       */
/*                                                                       */
/*************************************************************************/


#include "i2cexprt.h"
#include "i2cdrivr.h"
#include "reg8584.h"


static void NoInitErrorProc(void);

void (*masterProc)(void) = NoInitErrorProc;
void (*slaveProc)(void)  = NoInitErrorProc;

BYTE master;
BYTE intMask;



static void NoInitErrorProc(void)
/********************************
 * Input(s)    : none.
 * Output(s)   : none.
 * Returns     : none.
 * Description : ERROR: Master or slave handler called while not initialized
 *************************************************************************/
{
    CR_8584 = 0xC0 | intMask;                /* release bus NoAck          */
}
```

```
void MainStateHandler(void)
/*************************
 * Input(s)    : none.
 * Output(s)   : none.
 * Returns     : none.
 * Description : Main event handler for I2C.
 **************************************************************************/
{
    if (master)
        masterProc();                          /* Master Mode              */
    else
        slaveProc();                           /* Slave Mode               */
}


interrupt void I2C_Interrupt(void)
/********************************
 * Input(s)    : none.
 * Output(s)   : none.
 * Returns     : none.
 * Description : Interrupt handler for I2C.
 **************************************************************************/
{
    MainStateHandler();
}


void I2C_InstallInterrupt(BYTE vector)
/************************************
 * Input(s)    : vector       Interrupt vector for register S3
 * Output(s)   : none.
 * Returns     : none.
 * Description : Install and enable interrupt for I²C.
 **************************************************************************/
{
    CR_8584 = 0x10;                       /* write interrupt vector        */
    VR_8584 = vector;                     /* set vector number             */

    intMask = ENI_MASK;

    CR_8584 = ESO_MASK | ENI_MASK;        /* set serial interface ON       */
}
```

APPENDIX V I2CEXPRT.H

```
/***************************************************************************/
/*  Acronym           : I2C  (I2C Driver package for PCF8584)             */
/*  Name of module    : I2CEXPRT.H                                         */
/*  Creation date     : 1995-08-01                                         */
/*  Program language  : C                                                  */
/*  Name              : P.H. Seerden                                       */
/*                                                                         */
/*  Description       : This module consists a number of exported         */
/*                        declarations of the I2C driver package. Include  */
/*                        this module in your source file if you want to   */
/*                        make use of one of the interface functions of the */
/*                        package.                                         */
/*                                                                         */
/*        (C) Copyright 1995 Philips Semiconductors B.V.                  */
/*            Product Concept & Application Laboratory Eindhoven (PCALE)  */
/*            Eindhoven - The Netherlands                                  */
/*                                                                         */
/***************************************************************************/
/*                                                                         */
/*  History:                                                               */
/*                                                                         */
/*  95-08-01    P.H. Seerden    Initial version                           */
/*                                                                         */
/*                                                                         */
/***************************************************************************/

#define NULL             ((void *) 0)       /* a null pointer              */

typedef unsigned char    BYTE;
typedef unsigned short   WORD;
typedef unsigned long    LONG;


typedef struct
{
    BYTE   address;            /* slave address to sent/receive message     */
    BYTE   nrBytes;            /* number of bytes in message buffer         */
    BYTE   *buf;               /* pointer to application message buffer     */
} I2C_MESSAGE;


typedef struct
{
    BYTE        nrMessages;           /* number of message in one transfer */
    I2C_MESSAGE     **p_message;              /* pointer to pointer to message     */
} I2C_TRANSFER;


/***************************************************************************/
/*        E X P O R T E D   D A T A   D E C L A R A T I O N S             */
/***************************************************************************/

#define FALSE       0
#define TRUE        1

#define I2C_WR      0
#define I2C_RD      1
```

```
/**** Status Errors ****/

#define I2C_OK                    0        /* transfer ended No Errors       */
#define I2C_BUSY                  1        /* transfer busy                  */
#define I2C_ERR                   2        /* err: general error             */
#define I2C_NO_DATA               3        /* err: No data in block          */
#define I2C_NACK_ON_DATA          4        /* err: No ack on data            */
#define I2C_NACK_ON_ADDRESS       5        /* err: No ack on address         */
#define I2C_DEVICE_NOT_PRESENT    6        /* err: Device not present        */
#define I2C_ARBITRATION_LOST      7        /* err: Arbitration lost          */
#define I2C_TIME_OUT              8        /* err: Time out occurred         */
#define I2C_SLAVE_ERROR           9        /* err: Slave mode error          */
#define I2C_INIT_ERROR            10       /* err: Initialization (not done) */


/*****************************************************************************/
/*        I N T E R F A C E   F U N C T I O N   P R O T O T Y P E S          */
/*****************************************************************************/

extern void I2C_InitializeMaster(BYTE speed);
extern void I2C_InitializeSlave(BYTE slv, BYTE *buf, BYTE size, BYTE speed);
extern void I2C_InstallInterrupt(BYTE vector);
extern interrupt void I2C_Interrupt(void);

extern void I2C_Write(I2C_MESSAGE *msg);
extern void I2C_WriteRepWrite(I2C_MESSAGE *msg1, I2C_MESSAGE *msg2);
extern void I2C_WriteRepRead(I2C_MESSAGE *msg1, I2C_MESSAGE *msg2);
extern void I2C_Read(I2C_MESSAGE *msg);
extern void I2C_ReadRepRead(I2C_MESSAGE *msg1, I2C_MESSAGE *msg2);
extern void I2C_ReadRepWrite(I2C_MESSAGE *msg1, I2C_MESSAGE *msg2);
```

APPENDIX VII I2CDRIVR.H

```
/***************************************************************************/
/*  Acronym            : I2C  (I2C Driver package for PCF8584)             */
/*  Name of module     : I2CDRIVR.H                                        */
/*  Creation date      : 1995-08-01                                        */
/*  Program language   : C                                                 */
/*  Name               : P.H. Seerden                                      */
/*                                                                         */
/*  Description        : This module consists a number of 'local'          */
/*                         declarations of the I2C PCx8584 driver package. */
/*                                                                         */
/*        (C) Copyright 1995 Philips Semiconductors B.V.                   */
/*            Product Concept & Application Laboratory Eindhoven (PCALE)   */
/*            Eindhoven - The Netherlands                                  */
/*                                                                         */
/***************************************************************************/
/*                                                                         */
/*  History:                                                               */
/*                                                                         */
/*  95-08-01    P.H. Seerden    Initial version                            */
/*                                                                         */
/*                                                                         */
/***************************************************************************/


#define ST_IDLE             0
#define ST_SENDING          1
#define ST_AWAIT_ACK        2
#define ST_RECEIVING        3
#define ST_RECV_LAST        4


#define SLAVE_IDLE          0
#define SLAVE_TRX           1
#define SLAVE_RCV           2
#define SLAVE_LAST          3
#define SLAVE_READY         4


#define ACK_MASK            0x01
#define STO_MASK            0x02
#define STA_MASK            0x04
#define ENI_MASK            0x08
#define ESO_MASK            0x40

#define BB_MASK             0x01
#define LAB_MASK            0x02
#define AAS_MASK            0x04
#define LRB_MASK            0x08
#define BER_MASK            0x10
#define STS_MASK            0x20
#define PIN_MASK            0x80


extern BYTE master;
extern BYTE intMask;
```

APPENDIX VII REG8584.H

```
/***************************************************************************/
/*  Acronym            : GENERAL                                            */
/*  Name of module     : REG8584.H                                          */
/*  Creation date      : 1995-08-01                                         */
/*  Program language   : C                                                  */
/*  Name               : P.H. Seerden                                       */
/*                                                                          */
/*  Description        : Hardware register (I/O port) description file of  */
/*                         the PCx8584 I²C - bus controller, for use in C   */
/*                         programs.                                        */
/*                                                                          */
/*       !!!!  CHANGE ADDRESSES FOR OTHER APPLICATIONS  !!!!               */
/*                                                                          */
/*        Copyright (C) Philips Semiconductors B.V.                        */
/*            Product Concept & Application Laboratory Eindhoven (PCALE)   */
/*            Eindhoven - The Netherlands                                   */
/*                                                                          */
/***************************************************************************/
/*                                                                          */
/*  History:                                                                */
/*                                                                          */
/*  95-08-01    P.H. Seerden    Initial version                             */
/*                                                                          */
/*                                                                          */
/***************************************************************************/


#define BYTE_AT(x)   (*(unsigned char *)x)


/*       !!!!  CHANGE ADDRESSES FOR OTHER APPLICATIONS  !!!!               */


                                       /*                     ES0 ES1 ES2    */
#define AR_8584     BYTE_AT(0xFF0001)   /* Address Register    0   0   0     */
#define VR_8584     BYTE_AT(0xFF0001)   /* Vector Register     0   0   1     */
#define CL_8584     BYTE_AT(0xFF0001)   /* Clock Register      0   1   0     */
#define DR_8584     BYTE_AT(0xFF0001)   /* Data Register       1   0   0     */

#define CR_8584     BYTE_AT(0xFF0003)   /* Control Register    0   x   x     */
#define CS_8584     BYTE_AT(0xFF0003)   /* Cntrl/Status Reg    1   x   x     */
```

APPENDIX VIII EXAMPLE.C

```
/*************************************************************************/
/*  Acronym            : I2C  Inter IC bus                               */
/*  Name of module     : EXAMPLE.C                                       */
/*  Scope              : Application software                            */
/*  12nc               : xxxx xxx xxxx.x                                 */
/*  Creation date      : 1995-08-01                                      */
/*  Program language   : C                                               */
/*  Name               : P.H. Seerden                                    */
/*                                                                       */
/*                                                                       */
/*        (C) Copyright 1994 Philips Semiconductors B.V.                */
/*            Product Concept & Application Laboratory Eindhoven (PCALE)   */
/*            Eindhoven - The Netherlands                                */
/*                                                                       */
/*  All rights are reserved. Reproduction in whole or in part is        */
/*  prohibited without the written consent of the copyright owner.      */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/*  Description:                                                         */
/*                                                                       */
/*     I²C driver test, for PCx8584                                      */
/*                                                                       */
/*     Tested on MICROCORE 2 and I²C evaluation board type OM1016       */
/*                                                                       */
/*     - Read the time from the real time clock chip PCF8583.           */
/*     - Displays the time on LCD module PCF8577 and LED module SAA1064.   */
/*     - Reads keys from I/O expander PCF8574.                           */
/*     - Depending of pushed keys send tone to loudspeaker PCD3312.     */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/*  History:                                                             */
/*                                                                       */
/*  95-08-01    P.H. Seerden    Initial version                          */
/*                                                                       */
/*                                                                       */
/*************************************************************************/

#include <stdio.h>

#include "i2cexprt.h"


#define PCF8574_WR   0x7E                    /* i2c address I/O poort write */
#define PCF8574_RD   0x7F                    /* i2c address I/O poort read  */
#define PCD3312_WR   0x4A                    /* i2c address DTMF            */
#define SAA1064_WR   0x76                    /* i2c address 7 segm. Led     */
#define SAA1064_RD   0x77                    /* i2c address 7 segm. Led     */
#define PCF8577_WR   0x74                    /* i2c address 7 segm. LCD     */
#define PCF8583_WR   0xA2                    /* i2c address Clock           */
#define PCF8583_RD   0xA3                    /* i2c address Clock           */


#define LCDA         0x80                    /* LCD segment a               */
#define LCDB         0x40                    /* LCD segment b               */
#define LCDC         0x20                    /* LCD segment c               */
#define LCDD         0x10                    /* LCD segment d               */
#define LCDE         0x08                    /* LCD segment e               */
#define LCDF         0x04                    /* LCD segment f               */
#define LCDG         0x02                    /* LCD segment g               */
#define LCDDP        0x01                    /* LCD segment dp              */

#define LEDA         0x04                    /* LED segment a               */
#define LEDB         0x08                    /* LED segment b               */
#define LEDC         0x40                    /* LED segment c               */
#define LEDD         0x20                    /* LED segment d               */
#define LEDE         0x10                    /* LED segment e               */
#define LEDF         0x01                    /* LED segment f               */
#define LEDG         0x02                    /* LED segment g               */
#define LEDDP        0x80                    /* LED segment dp              */
```

```
const BYTE lcdTbl[] = { LCDA+LCDB+LCDC+LCDD+LCDE+LCDF,      /*  0         */
                        LCDB+LCDC,                           /*  1         */
                        LCDA+LCDB+LCDG+LCDD+LCDE,            /*  2         */
                        LCDA+LCDB+LCDG+LCDD+LCDC,            /*  3         */
                        LCDB+LCDG+LCDF+LCDC,                 /*  4         */
                        LCDA+LCDF+LCDG+LCDC+LCDD,            /*  5         */
                        LCDA+LCDF+LCDG+LCDC+LCDD+LCDE,       /*  6         */
                        LCDA+LCDB+LCDC,                      /*  7         */
                        LCDA+LCDB+LCDC+LCDD+LCDE+LCDF+LCDG,  /*  8         */
                        LCDA+LCDB+LCDC+LCDD+LCDF+LCDG,       /*  9         */
                        LCDA+LCDB+LCDC+LCDE+LCDF+LCDG,       /*  A         */
                        0,                                   /*  blank     */
                        LCDA,                                /*            */
                        LCDB+LCDC+LCDD+LCDE+LCDG,            /*  d         */
                        LCDG,                                /*  e         */
                        LCDD,                                /*            */
                        LCDDP                                /*            */
                      };

const BYTE ledTbl[] = { LEDA+LEDB+LEDC+LEDD+LEDE+LEDF,      /*  0         */
                        LEDB+LEDC,                           /*  1         */
                        LEDA+LEDB+LEDG+LEDD+LEDE,            /*  2         */
                        LEDA+LEDB+LEDG+LEDD+LEDC,            /*  3         */
                        LEDB+LEDG+LEDF+LEDC,                 /*  4         */
                        LEDA+LEDF+LEDG+LEDC+LEDD,            /*  5         */
                        LEDA+LEDF+LEDG+LEDC+LEDD+LEDE,       /*  6         */
                        LEDA+LEDB+LEDC,                      /*  7         */
                        LEDA+LEDB+LEDC+LEDD+LEDE+LEDF+LEDG,  /*  8         */
                        LEDA+LEDB+LEDC+LEDD+LEDF+LEDG,       /*  9         */
                        LEDA+LEDB+LEDC+LEDE+LEDF+LEDG,       /*  A         */
                        0,                                   /*  blank     */
                        LEDA,                                /*            */
                        LEDB+LEDC+LEDD+LEDE+LEDG,            /*  d         */
                        LEDG,                                /*  e         */
                        LEDD,                                /*            */
                        LEDDP                                /*            */
                      };


static BYTE  ledBuf[5];
static BYTE  lcdBuf[5];
static BYTE  rtcBuf[4];
static BYTE  iopBuf[1];
static BYTE  sndBuf[1];


static I2C_MESSAGE  rtcMsg1;
static I2C_MESSAGE  rtcMsg2;
static I2C_MESSAGE  iopMsg;
static I2C_MESSAGE  sndMsg;
static I2C_MESSAGE  ledMsg;
static I2C_MESSAGE  lcdMsg;
```

```
static void Init(void)
{
  void   **ptr;
  #define VECTOR_BASE   0x40000             /* start of vector table       */

    ptr  = (void *) (VECTOR_BASE + (4 * 28));
    *ptr = (void *) I2C_Interrupt;

    I2C_InitializeMaster(0x10);           /* 4.43MHz and SCL = 90Khz       */

    I2C_InstallInterrupt(28);             /* Interrupt vector number       */

    ledMsg.address = SAA1064_WR;
    ledMsg.nrBytes = 2;
    ledMsg.buf     = ledBuf;
    ledBuf[0] = 0;
    ledBuf[1] = 0x47;
    I2C_Write(&ledMsg);                     /* led brightness              */
    ledMsg.nrBytes = 5;

    rtcBuf[0] = 2;                            /* sub address               */
    rtcBuf[1] = 0x00;                         /* seconds                   */
    rtcBuf[2] = 0x59;                         /* minutes                   */
    rtcBuf[3] = 0x23;                         /* hours                     */
    rtcMsg1.address = PCF8583_WR;
    rtcMsg1.nrBytes = 4;
    rtcMsg1.buf     = rtcBuf;
    I2C_Write(&rtcMsg1);                      /* set clock                 */

    rtcBuf[0] = 2;                            /* sub address               */
    rtcMsg1.nrBytes = 1;
    rtcMsg1.buf     = rtcBuf;
    rtcMsg2.address = PCF8583_RD;
    rtcMsg2.nrBytes = 3;
    rtcMsg2.buf     = rtcBuf;

    iopMsg.address = PCF8574_RD;
    iopMsg.buf     = iopBuf;
    iopMsg.nrBytes = 1;

    sndMsg.address = PCD3312_WR;
    sndMsg.buf     = sndBuf;
    sndMsg.nrBytes = 1;

    lcdMsg.address = PCF8577_WR;
    lcdMsg.buf     = lcdBuf;
    lcdMsg.nrBytes = 5;
}


static void HandleKeys(void)
{
    I2C_Read(&iopMsg);

    switch ((iopBuf[0] ^ 0xFF) & 0x0F)
    {
      case 0  : sndBuf[0] = 0x01;    break;
      case 1  : sndBuf[0] = 0x30;    break;
      case 2  : sndBuf[0] = 0x31;    break;
      case 3  : sndBuf[0] = 0x32;    break;
      case 4  : sndBuf[0] = 0x33;    break;
      case 5  : sndBuf[0] = 0x34;    break;
      case 6  : sndBuf[0] = 0x35;    break;
      case 7  : sndBuf[0] = 0x36;    break;
      case 8  : sndBuf[0] = 0x37;    break;
      case 9  : sndBuf[0] = 0x38;    break;
      case 10 : sndBuf[0] = 0x39;    break;
      case 11 : sndBuf[0] = 0x3A;    break;
      case 12 : sndBuf[0] = 0x29;    break;
      case 13 : sndBuf[0] = 0x3B;    break;
      case 14 : sndBuf[0] = 0x3C;    break;
      case 15 : sndBuf[0] = 0x3D;    break;
    }
    I2C_Write(&sndMsg);
}
```

```
void main(void)
{
  BYTE  oldseconds = 0;

    Init();

    ua_init();    /* init uart used for I/O redirection in printf()        */

    while (1)
    {
        HandleKeys();

        rtcBuf[0] = 2;                                 /* sub address         */
        I2C_WriteRepRead(&rtcMsg1, &rtcMsg2);

        if (rtcBuf[0] != oldseconds)      /* check if one second is passed */
        {
            oldseconds = rtcBuf[0];

            lcdBuf[0] = 0;
            if (oldseconds & 1)
                lcdBuf[1] = lcdTbl[rtcBuf[2] >> 4];
            else
                lcdBuf[1] = lcdTbl[rtcBuf[2] >> 4] | LCDDP;
            lcdBuf[2] = lcdTbl[rtcBuf[2] & 0x0F];
            lcdBuf[3] = lcdTbl[rtcBuf[1] >> 4];
            lcdBuf[4] = lcdTbl[rtcBuf[1] & 0x0F];
            I2C_Write(&lcdMsg);

            ledBuf[0] = 1;
            ledBuf[1] = 2;
            ledBuf[2] = ledTbl[rtcBuf[0] >> 4];
            ledBuf[3] = ledTbl[rtcBuf[0] & 0x0F];
            ledBuf[4] = 2;
            I2C_Write(&ledMsg);
        }
    }
}
```