APPLICATION NOTE

XA-S3 I2C driver software

AN98046

Philips
Semiconductors

PHILIPS

XA-S3 I2C driver software

Application Note
AN98046

Abstract

The XA-S3 is a member of Philips Semiconductors' XA (eXtended Architecture) family of high performance 16-bit single-chip Microcontrollers. The XA-S3 combines many powerful peripherals on one chip. Therefore, it is suited for general multipurpose high performance embedded control functions.

One of the on-chip peripherals is the I2C bus interface. This report describes worked-out driver software (written in C) to program / use the I2C interface of the XA-S3. The driver software, together with a demo program and interface software routines offer the user a quick start in writing a complete I2C - XAS3 system application.

Purchase of Philips I^2C components conveys a license under the I^2C patent to use the components in the I^2C system, provided the system conforms to the I^2C specifications defined by Philips.

APPLICATION NOTE

XA-S3 I2C driver software

AN98046

Author(s):

Paul Seerden
Philips Semiconductors Systems Laboratory Eindhoven,
The Netherlands

Keywords

XA Microcontroller, I2C - bus, Application software

Number of pages: 14

Date: 98-04-02

XA-S3 I2C driver software

Application Note AN98046

Summary

This application note demonstrates how to write an Inter Integrated Circuit bus driver (I^2C) for the XA-S3 16-bit Microcontroller from Philips Semiconductors.

Not only the driver software is given. This note also contains a set of (example) interface routines and a small demo application program. All together it offers the user a quick start in writing a complete I^2C system application with the PXAS3x.

The driver routines support interrupt driven single master transfers. Furthermore, the routines are suitable for use in conjunction with real time operating systems.

CONTENTS

1. INTRODUCTION 7

2. EXTERNAL (APPLICATION) INTERFACE 9

2.1 External data interface 9
2.2 External function interfaces 10

3. DRIVER OPERATION 11

4. DEMO PROGRAM 12

APPENDIX 1 I2CINTFC.C 13

APPENDIX 2 I2CDRIVR.C 17

APPENDIX 3 I2CEXPRT.H 20

APPENDIX 4 DEMO.C 22

1. INTRODUCTION

This report describes I^2C driver software, written in C, for the XA-S3 Microcontroller. The driver software is the interface between application software and the (hardware) 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 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. However, the driver software given in this application note only supports (single) master transfers.

The I²C interface on the XA-S3 is identical to the standard byte - style I²C interface found on devices such as the 8xC552, except for the rate selection. The I²C interface conforms to the 100 kHz I2C specification, but may be used at rates up to 400 kHz (non-conforming).

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 slave may stretch clock pulses (for timing causes).

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

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.

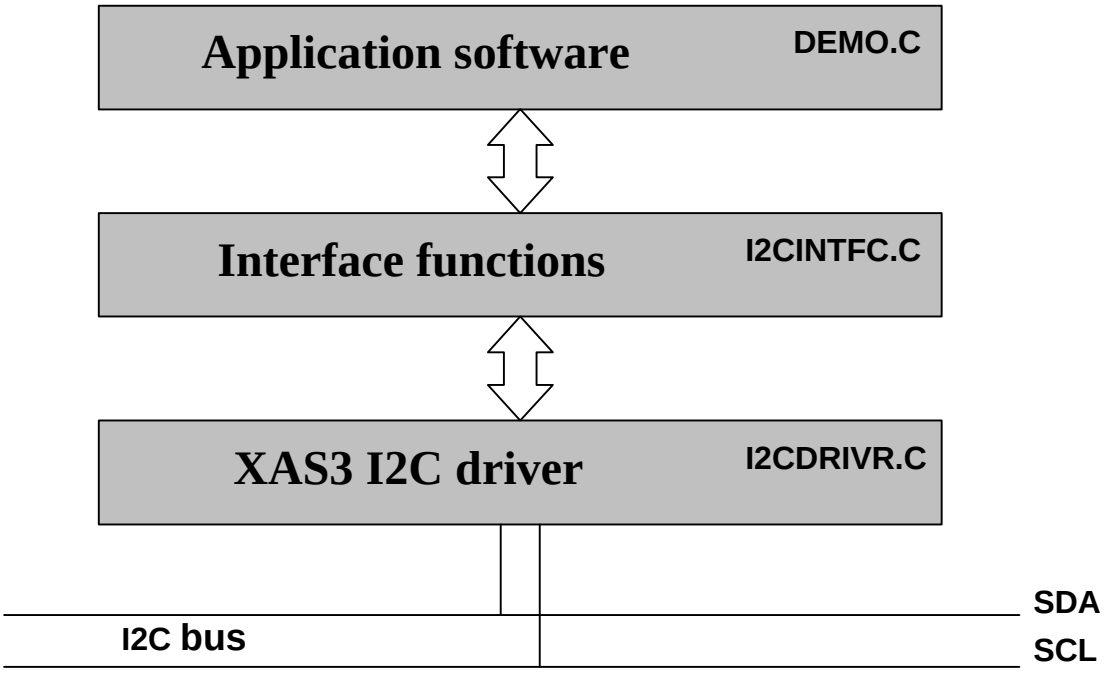
D1 ... Dn : Block of data bytes.

Also:

D1.1 ... D1.m : First block of data bytes.

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

Software structure and modules



2. EXTERNAL (APPLICATION) INTERFACE

This section chapter describes 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.

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 message: the data bytes to be written to the slave.

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

⇒ Status information (success or error code).

⇒ Number of messages actually transferred (not the requested number of messages in case of an error).

⇒ For each read message: The data bytes read from the slave.

2.1 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 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:

```
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 lowest bit of the slave address determines the direction of the transfer (read or write);

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 driver fills the array. If you want to use ***buf*** as a string, a terminating NULL should be added at the end. It is the user's responsibility to ensure that the buffer, pointed to by ***buf*****,** is large enough to receive ***nrBytes*** bytes.

2.2 External function interfaces

This section gives a description of the two 'callable' interface functions in the I²C driver module (I2CDRIVR.C).

First the initialisation function (*I2C_Initialize*) is explained. This function directly programs the I²C interface hardware and is part of the low level driver software. It must be called only once after 'reset', but before any transfer function is executed. After that the interface function, used to actually perform a transfer (*I2C_Transfer*), is explained.

void I2C_Initialize(void)

Initialise the I²C-bus driver part. Must be called once after RESET.

Hardware I²C registers of the XA-S3 will be programmed. The interrupt vector and the priority of the I²C interrupt are set. Used constants (parameters) are defined in the file I2CEXPRT.H. The port pins P5.6 and P5.7, which correspond to the I²C functions SCL and SDA respectively, are set to the open drain mode. The listed driver (see appendix 2) programs the bit rate to 80 Kbit/s at an oscillator frequency of 22.1184 KHz. To adapt this, change the I2CON values at the top of the file (check also register SCR for the peripheral clock pre-scaler).

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_TIME_OUT	err: Time out occurred

3. DRIVER OPERATION

The XA-S3 on-chip logic provides a serial interface that meets the I^2C bus specification and supports all transfer modes from and to the bus. The I^2C logic interfaces to the external I^2C bus via two port pins: P5.6/SCL (serial clock line) and P5.7/SDA (serial data line). In order to enable the interface these port pins are programmed to their alternate function and are set to open drain I/O port mode.

The XA processor interfaces to the I^2C logic via four hardware registers: I2CON (control register), I2STA (status register), I2DAT (data register) and I2ADR (slave address registers).

If a transfer is started, the drivers interface function returns immediately. At the end of **the transfer**, together with the generation of a STOP condition, the driver calls a function *(readyProc)*, passing the transfer status. This status (error, time-out, etc.) must be checked by the application. An example of how to handle the status is shown in the file I2CINTFC.C. A pointer to the *readyProc* function was given by the application at the time the transfer was applied for (see previous chapter).

After completing the transmission or reception **of each byte** (address or data), the SI flag in the I2CON register is set. An interrupt is requested and the interrupt service handler will be called. At that time register I2STA holds one of the following status codes (only master mode):

Master transmitter:

- 08H - A start condition has been transmitted
- 10H - A repeated start condition has been transmitted
- 18H - SLV_W has been transmitted, ACK received
- 20H - SLV_W has been transmitted, NOTACK received
- 28H - DATA from SDAT has been transmitted, ACK received
- 30H - DATA from SDAT has been transmitted, NOTACK received
- 38H - Arbitration lost in SLV_ R/W or DATA

Master receiver:

- 38H - Arbitration lost while returning NOTACK
- 40H - SLV_R has been transmitted, ACK received
- 48H - SLV_R has been transmitted, NOTACK received
- 50H - DATA in SDAT received, ACK returned
- 58H - DATA in SDAT received, NOTACK returned

Miscellaneous:

- 00H - Bus error during master or selected slave mode, due to an erroneous START or STOP condition.

4. DEMO PROGRAM

The modules DEMO.C and I2CINTFC.C use the driver to implement a simple application on a Microcore 7 demo / evaluation board. They are intended as examples to show how to use the driver routines.

The Microcore 7 board contains a PCF8574A I/O expander with connections to 8 LED's. The demo program runs the LED's every second.

The module I2CINTFC.C gives an example of how to implement a few basic transfer functions (see also previous SLE 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. If a transfer has failed (error or time-out) the *StartTransfer* function prints an error message (using standard I/O redirection, like the *printf()* function) and it does a retry of the transfer. However, if the maximum number of retries is reached an exception interrupt (Trap #14) is generated to give a fatal error message.

APPENDIX 1 I2CINTFC.C

```
/*****************************************************************************/
/*  Name of module : I2CINTFC.C                                              */
/*  Language       : C (Hi-Tech XA compiler V7.70)                           */
/*  Name           : P.H. Seerden                                            */
/*  Description    : External interface to the XA-S3 I2C driver              */
/*                   routines. This module contains the **EXAMPLE**          */
/*                   interface functions, used by the application to         */
/*                   do I2C master-mode transfers.                           */
/*                                                                           */
/*           (C) Copyright 1998 Philips Semiconductors B.V.                  */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/*  History:                                                                 */
/*                                                                           */
/*  97-10-29    P.H. Seerden    Initial version                              */
/*  98-03-30    P.H. Seerden    Updated version                              */
/*                                                                           */
/*****************************************************************************/

#include "i2cexprt.h"



extern void PrintString(code char *s);

code char retryexp[] = "retry counter expired\n";
code char bufempty[] = "buffer empty\n";
code char nackdata[] = "no ack on data\n";
code char nackaddr[] = "no ack on address\n";
code char timedout[] = "time-out\n";
code char unknowst[] = "unknown status\n";


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 (retries == 6)
        {
            PrintString(retryexp);         /* fatal error ! So, ..         */
            asm("trap  #14");              /* escape to debug monitor      */
        }
        else
            retries++;

        switch (drvStatus)
        {
          case I2C_OK                : break;
          case I2C_NO_DATA           : PrintString(bufempty);    break;
          case I2C_NACK_ON_DATA      : PrintString(nackdata);    break;
          case I2C_NACK_ON_ADDRESS   : PrintString(nackaddr);    break;
          case I2C_TIME_OUT          : PrintString(timedout);    break;
          default                    : PrintString(unknowst);    break;
        }
    } while (drvStatus != I2C_OK);
}



void I2C_Write(I2C_MESSAGE *msg)
/*****************************
 * Input(s)   : msg     IýC 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 2 I2CDRIVR.C

```
/*************************************************************************/
/*  Name of module   : I2CDRIVR.C                                        */
/*  Program language : C (Hi-Tech XA compiler V7.70)                     */
/*  Name             : P.H. Seerden                                      */
/*  Description      : Driver for the I2C hardware interface on the      */
/*                     Philips XA-S3 16-bit microcontroller.             */
/*                     Part of the driver handling master bus transfers. */
/*                     Everything between a Start and Stop condition is  */
/*                     called a TRANSFER. One transfer consists of one or*/
/*                     more MESSAGEs. To start a transfer call function  */
/*                     "I2C_Transfer".                                   */
/*                                                                       */
/*           (C) Copyright 1998 Philips Semiconductors B.V.              */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/*  History:                                                             */
/*                                                                       */
/*  97-10-28    P.H. Seerden    Initial version                          */
/*  98-03-30    P.H. Seerden    Updated version                          */
/*                                                                       */
/*************************************************************************/

#include <xas3.h>
#include <intrpt.h>
#include "i2cexprt.h"

/*    Immediate data to write into I2CON                                 */
/*    CR2-CR1-CR0 = 101 (80KHz I2C bitrate at 22.1184 MHz crystal freq)  */
/*    change values and recompile if a different bus speed is needed     */

#define GENERATE_STOP        0xD5     /* set STO, clear STA and SI       */
#define RELEASE_BUS_ACK      0xC5     /* clear STO,STA,SI and set AA (ack) */
#define RELEASE_BUS_NOACK    0xC1     /* clear STO, STA, SI and AA (noack) */
#define RELEASE_BUS_STA      0xE5     /* generate (rep)START, set STA    */

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 msgCount;                 /* Number of messages to sent      */
static BYTE dataCount;                /* bytes send/received of message  */


interrupt void I2C_Interrupt(void)
/*******************************/
{
    switch(I2STAT)                       /* >> 3 for faster code         */
    {
      case 0x00:                         /* Bus Error has occured        */
        I2CON = GENERATE_STOP;
        break;
      case 0x08:                   /* (rep) Start condition transmitted  */
      case 0x10:                   /* Slave address + R/W are transmitted */
        I2DAT = msg->address;
        I2CON = RELEASE_BUS_ACK;
        break;
      case 0x18:               /* SLA+W or DATA transmitted, ACK received */
      case 0x28:               /* DATA or STOP will be transmitted       */
        if (dataCount < msg->nrBytes)
        {
            I2DAT = msg->buf[dataCount++];          /* sent first byte   */
            I2CON = RELEASE_BUS_ACK;
        }
        else
        {
            if (msgCount < tfr->nrMessages)
            {
```

```
                dataCount = 0;
                msg = tfr->p_message[msgCount++];   /* next message        */
                I2CON = RELEASE_BUS_STA;            /* generate (rep)START */
            }
            else
            {
                I2CON = GENERATE_STOP;
                readyProc(I2C_OK, msgCount);
            }
          }
          break;
        case 0x20:
        case 0x48:              /* SLA+W/R transmitted, NOT ACK received     */
          readyProc(I2C_NACK_ON_ADDRESS, msgCount);      /* driver finished  */
          I2CON = GENERATE_STOP;
          break;
        case 0x30:              /* DATA transmitted, NOT ACK received        */
          readyProc(I2C_NACK_ON_DATA, msgCount);
          I2CON = GENERATE_STOP;
          break;
        case 0x38:              /* Arbitration lost in SLA+W or DATA         */
          I2CON = RELEASE_BUS_STA;              /* release bus, set STA    */
          break;
        case 0x40:                      /* SLA+R transmitted, ACK received   */
          if (msg->nrBytes == 1)
              I2CON = RELEASE_BUS_NOACK;        /* No ack on next byte     */
          else
              I2CON = RELEASE_BUS_ACK;          /* ACK on next byte        */
          break;
        case 0x50:                /* DATA received, ACK has been returned    */
          msg->buf[dataCount++] = I2DAT;        /* read next data          */
          if (dataCount + 1 == msg->nrBytes)    /* next byte the last ?    */
              I2CON = RELEASE_BUS_NOACK;        /* No ack on next byte     */
          else
              I2CON = RELEASE_BUS_ACK;          /* return ACK              */
          break;
        case 0x58:              /* DATA received, NOT ACK has been returned  */
          msg->buf[dataCount] = I2DAT;          /* read last data          */
          if (msgCount < tfr->nrMessages)
          {
              dataCount = 0;
              msg = tfr->p_message[msgCount++];     /* next message          */
              I2CON = RELEASE_BUS_STA;              /* generate (rep)START   */
          }
          else
          {
              I2CON = GENERATE_STOP;
              readyProc(I2C_OK, msgCount);
          }
          break;
        default: break;
    }
}


void I2C_Initialize(void)
/***********************/
{
    ROM_VECTOR(0xd4,I2C_Interrupt,IV_PSW);      /* I2C interrupt vector    */

    P5CFGA = P5CFGA & 0x3f;       /* P5.6 and P5.7 as open drain ports     */
    P5CFGB = P5CFGB & 0x3f;

    I2ADDR = 0x26;                /* set default slave address             */
    I2CON  = RELEASE_BUS_ACK;     /* set speed and enable IýC hardware     */
    IPB2 = IPB2 | 0x10;           /* set priority of I2C interrupt to 9    */
    EI2 = 1;                      /* enable I2C interrupt                  */
    EA  = 1;                      /* General interrupt enable              */
}



void I2C_Transfer(I2C_TRANSFER *p, void (*proc)(BYTE, BYTE))
```

```
/*********************************************************/
{
    tfr = p;
    readyProc = proc;
    msgCount  = 0;
    dataCount = 0;
    msg = tfr->p_message[msgCount++];     /* first message to send         */

    I2CON = RELEASE_BUS_STA;              /* generate START condition      */
}
```

APPENDIX 3 I2CEXPRT.H

```
/*************************************************************************/
/*  Name of module    : I2CEXPRT.H                                       */
/*  Program language  : C                                                */
/*  Name              : P.H. Seerden                                     */
/*                                                                       */
/*        (C) Copyright 1998 Philips Semiconductors B.V.                 */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/*  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.        */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/*  History:                                                             */
/*                                                                       */
/*  92-12-10    P.H. Seerden    Initial version                          */
/*                                                                       */
/*************************************************************************/

#define FALSE       0
#define TRUE        1

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;


#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) */



extern void I2C_Transfer(I2C_TRANSFER *p, void (*proc)(BYTE, BYTE));
extern void I2C_Initialize(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 4 DEMO.C

```
/*****************************************************************************/
/*  Name of module    : DEMO.C                                               */
/*  Program language  : C                                                    */
/*  Name              : P.H. Seerden                                         */
/*  Description       : XA-S3 I2C driver test                                */
/*                      Read time from the real time clock chip PCF8583.     */
/*                      run leds connected to PCF8574 every second.          */
/*                                                                           */
/*****************************************************************************/

#include "i2cexprt.h"

#define PCF8574_WR   0x40                  /* i2c address I/O poort write */
#define PCF8583_WR   0xA0                  /* i2c address Clock           */
#define PCF8583_RD   0xA1                  /* i2c address Clock           */

static BYTE  rtcBuf[1];
static BYTE  iopBuf[1];
static I2C_MESSAGE  rtcMsg1;
static I2C_MESSAGE  rtcMsg2;
static I2C_MESSAGE  iopMsg;


static void Init(void)
{
    I2C_Initialize();

    rtcMsg1.address = PCF8583_WR;
    rtcMsg1.buf     = rtcBuf;
    rtcMsg1.nrBytes = 1;
    rtcMsg2.address = PCF8583_RD;
    rtcMsg2.buf     = rtcBuf;
    rtcMsg2.nrBytes = 1;

    iopMsg.address = PCF8574_WR;
    iopMsg.buf     = iopBuf;
    iopMsg.nrBytes = 1;
    iopBuf[0] = 0xff;
    I2C_Write(&iopMsg);
}

void main(void)
{
  BYTE  oldseconds,port;

    Init();
    oldseconds = 0;
    port = 0xf7;
    while (1)
    {
        rtcBuf[0] = 2;                          /* read seconds          */
        I2C_WriteRepRead(&rtcMsg1, &rtcMsg2);
        if (rtcBuf[0] != oldseconds)            /* one second passed ?   */
        {
            oldseconds = rtcBuf[0];
            switch (port)
            {
              case 0xf7: port = 0xfe;   break;
              case 0xfb: port = 0xf7;   break;
              case 0xfd: port = 0xfb;   break;
              case 0xfe: port = 0xfd;   break;
              default:  break;
            }
            iopBuf[0] = port;
            I2C_Write(&iopMsg);
        }
    }
}
```