www.DataSheet4U.com

APPLICATION NOTE

DataSheet4U.com

DataShee

AN96119
I²C with the XA-G3

Paul Seerden,
Systems Laboratory Eindhoven, The Netherlands

January 1997

DataSheet4U.com

Philips
Semiconductors

www.DataSheet4U.com

PHILIPS

DataSheet4U.com

DataSheet4U.com

www.DataSheet4U.com

I²C with the XA-G3 AN96119

Author: Paul Seerden, Systems Laboratory Eindhoven, The Netherlands

ABSTRACT

This report describes how to implement I²C functionality (single master), if you're using the Philips XA-Gx microcontroller. Elaborated driver routines (written in C) are given for two alternative solutions:

– Software emulation using two port pins ('bit-banging').

– Using the PCx8584 I²C-bus controller.

SUMMARY

This application note demonstrates the implementation of I²C functionality using the 16-bit XA-G3 microcontroller from Philips Semiconductors.

The note contains two main parts:

– An implementation using the Philips PCx8584 I²C-bus controller (Interrupt driven).

– An implemenation by software emulation of the bus using 2 I/O port pins (polling, 'bit-banging').

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 I²C system application (single master).

Purchase of Philips I²C components conveys a license under the Philips' I²C patent to use the components in the I²C system provided the system conforms to the I²C specifications defined by Philips. This specification can be ordered using the code 9398 393 40011.

DataSheet4U.com

CONTENTS

DataSheet4U.com
DataSheet4U.com
DataSheet4U.com
www.DataSheet4U.com

www.DataSheet4U.com

1. INTRODUCTION

This report describes I2C driver software, in C, for the XA microcontroller. This 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 adddress. 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 bnus 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.

Chapter 2 gives a functional description of the driver program.

Chapter 3 describes the software structure and all driver interface functions (‘callable’ by the application).

Chapter 4 describes the low level hardware dependent driver software and is split into one general part and two sections. The first section describes a software emulated I2C bus driver (‘bit-banging’) using two I/O port pins. The other section describes an interrupt driven PCF8584 driver. The PCF8584 is a Philips integrated circuit to be used as a separate I2C bus controller.

Chapter 5 is a short description of the example application program (demo.c and i2cintfc.c).

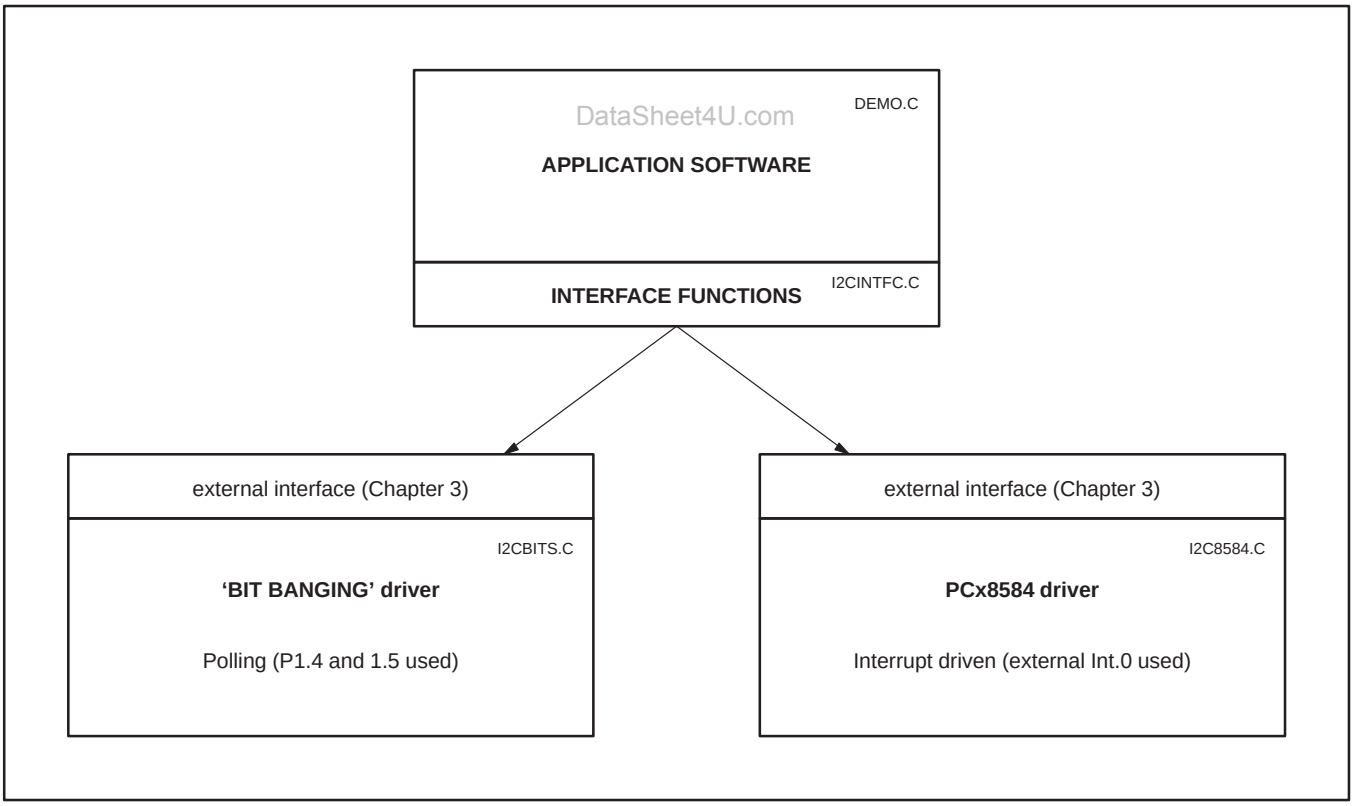


Figure 1. Overview of software layers and modules

DataSheet4U.com
www.DataSheet4U.com

DataSheet4U.com
DataSheet4U.com

www.DataSheet4U.com

1.1 References

Description	Ordering info
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	see BBS / WWW
Specification I²C driver (J. Reitsma)	
C routines for the PCx8584	AN95068
I2CBITS.ASM (by G. Goodhue)	see BBS / WWW
Used development and test tools:	
Hi-Tech C XA complier (version 7.60)	http://www.htsoft.com
Philips Microcore SiXA evaluation board	
FDI XTEND board	XTEND-G3
Philips I²C-bus evaluation board	OM1016
Philips Logic Analyzer with I²C-bus support package PF8681	PM3580/PM3585

et4U.com DataShee

1.2 BBS and WWW

DataSheet4U.com

This application note (with C source files) is available for downloading from the Philips Bulletin Board Systems and from the world wide web. It is packed in the self extracting PC DOS file: I2CXAG3.EXE.

To better serve our customers, Philips maintains a microcontroller bulletin board. This system is open to all callers, it operates 24 hours a day, and can be accessed with modems up to 28800 bps. The telephone number is:

European Bulletin Board, telephone number: +31 40 272 1102.

Internet access:

Philips Semiconductors WWW: http://www.semiconductors.philips.com

1.3 File overview

the driver package contains the following files:

I2CBITS.C The driver part for ‘bit-banging” I²C.

I2C8584.C The PCF8584 drive for master transfers, containing initialization and state handling. This module also contains address definitions of hardware registers of the PCx8584. The user should adapt these definitions to his own system environment (address map).

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 handlin, like: time-outs (software loops), retries and error messages. The user must adapt these funtions to his own system needs and environment.

I2CEXPRT.H This module (include file) contains definitions of all ‘global’ contants, function prototypes, data tyes and structures needed by the user (application). Include this file in the user application source files.

DataSheet4U.com www.DataSheet4U.com

DataSheet4U.com DataSheet4U.com

www.DataSheet4U.com

DEMO.C This program uses the driver package to implement a simple application on the Microcore 6 demo / evaluation board. This board contains a PCx8584 I²C-bus controller, a PCF8583 real time clock and a PCF8574 I/O exapander with connections to 4 LEDs. The program runs the LEDs every second.

All driver software programs are tested as thoroughly as time permitted; however, Philips cannot guarantee that they are flawless in all applications.

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, follwing the data byte, the receiver must send an acknowledge bit to the transmitter. The clock speed is normally 100kHz. Clock pulses may be stretched (for timing causes) by the slave.

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

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

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

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

2.2 Input definition

Inputs (applicaiton'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.

DataSheet4U.com
DataSheet4U.com
www.DataSheet4U.com
DataSheet4U.com

www.DataSheet4U.com

2.3 Output definition

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.

2.4 Performance

The default maximum speed of the I²C-bus is 100 KHz. With the XA-Gx running at 16 MHz or higher, it's possible to reach this speed using the 'bit-banging' driver. However, it is important to minimize the delay time between successive data bytes, because this delay determines the effective speed of the bus.

The maximum speed of the PCF8584 is limited to 90 KHz.

Software emulation ('bit-banging') of the bus is a heavy load for the XA processor. That's why systems that have to do more time critical tasks better apply the interrupt driven PCF8584 solution.

2.5 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 hardware timers) are shown in the file I2CINTFC.C

2.6 Hardware requirements

Bit-bang driver:

The bus requires open-drain device outputs to drive the bus. In fact, all port pins of the XA-Gx are programmable to open-drain outputs. In our example, external memory is connected to port P0 and P2. So, we have chosen to use P1.4 as SDA pin and P1.5 as SCL pin. To change this (for example to P1.6 and P1.7, like at the C51 derivative), adjust the include file I2CDRIVR.H. The code size of the emulation driver in this application note is approximately 800 bytes (Hi-Tech compiler V7.60). The driver is tested and tuned for an XA-G3 running at 20 MHz.

PCx8584 driver:

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). The driver uses external interrupt 0 input. To change the base-address (0xF0000) of the PCF8584 edit the file I2C8584.C. In our example, a 3.6864 MHz clock is connected to the PCF8584.

DataSheet4U.com www.DataSheet4U.com

DataSheet4U.com DataSheet4U.com

www.DataSheet4U.com

3. EXTERNAL (APPLICATION) INTERFACE

This 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 applicaiton's view on the I²C bus is quite simple: The applicaiton can send messages to an I²C device. Also, the applicaiton 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 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.

et4U.com DataShee

The structure contains a pointer (p_message) to an array with pointers to the structure I2C_MESSAGE:

DataSheet4U.com

```
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 user's responsibility to ensure that the buffer, pointed to by ***buf***, is large enough to receive ***nrBytes*** bytes.

3.2 External function interfaces

This seciton gives a description of the only two 'callable' interface funcitons in the both I²C driver modules.

First, the initialization 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.

DataSheet4U.com www.DataSheet4U.com

DataSheet4U.com
DataSheet4U.com

www.DataSheet4U.com

void I2C_Initialize(BYTE speed)
Initialize the I²C-bus driver part. Must be called once after RESET.

'Bit Bang': Port pins P1.3 (SCL) and P1.4 (SDA) are programmed to be used as open-drain output pins.
BYTE **speed** Dummy parameter. Not used.

PCx8584: Hardware I²C registers of the PCx8584 interface will be programmed.
Used constants (parameters) are defined in the file I2CDRIVR.H.
BYTE **speed** Contents for clock register S2 (bit rate of I²C-bus).

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

et4U.com DataShee DataSheet4U.com DataSheet4U.com www.DataSheet4U.com

DataSheet4U.com DataSheet4U.com

www.DataSheet4U.com

4. DRIVER OPERATION

After completing a transfer the function *readyProc* in the application (or interface) is called.

After completing the transmission or reception of each byte (address or data), a state handler is called, either by interrupt (PCx8584) or by software ('bit-banging'). This handler can be in one of the following states:

ST_IDLE	The state handler does not expect any bus activity.
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.

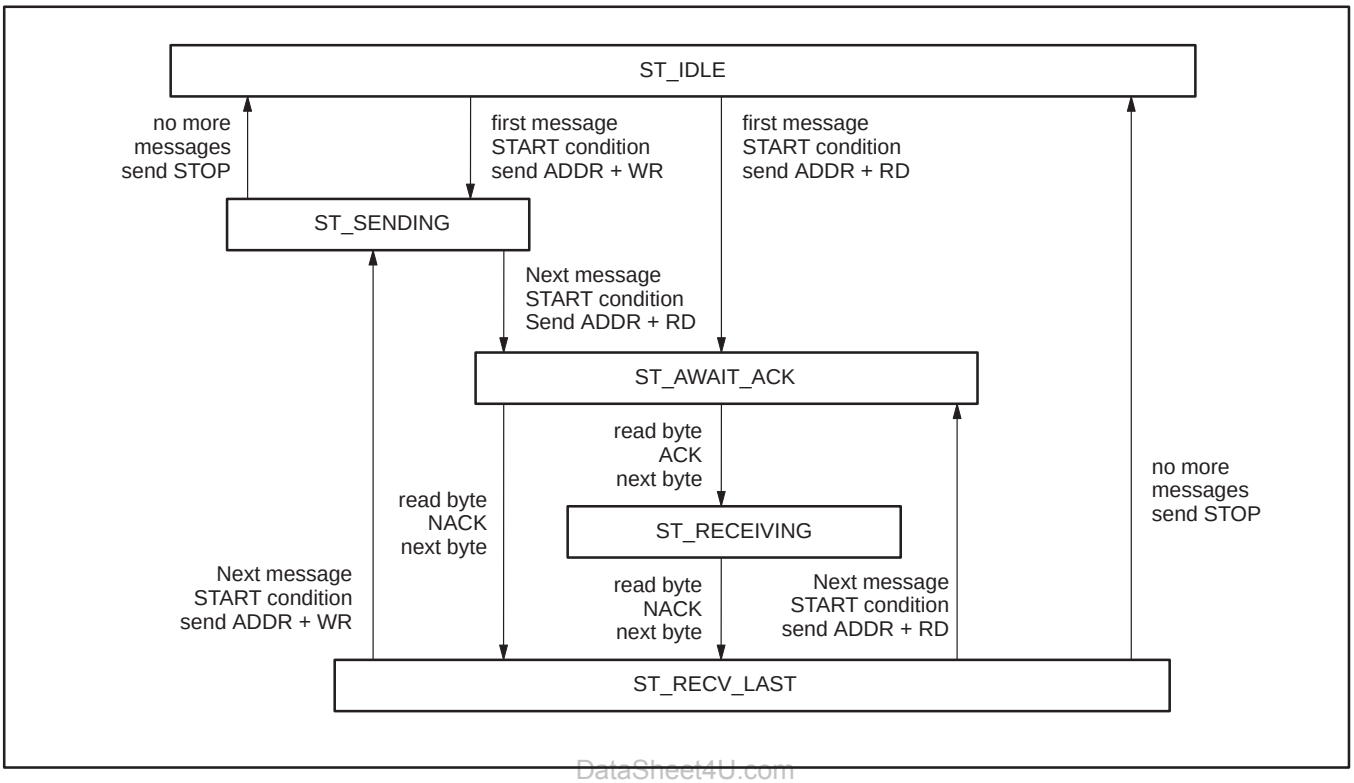
Figure 2 shows the state transition diagram. A transition will occur on initiation of a transfer by the application and on each I²C-bus event (state change). The transitions are:

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 send. Send the next byte. Or no more bytes to send, send repeated start and slave address of next message to write.
ST_SENDING → ST_IDLE	No more bytes to send, no more messages.
ST_SENDING → ST_AWAIT_ACK	No more bytes to send, 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. Call ReadyProc and give status.
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.

DataSheet4U.com
www.DataSheet4U.com

DataSheet4U.com
DataSheet4U.com

www.DataSheet4U.com



DataSheet4U.com

Figure 2. State transition diagram of the master state handler

4.1 Bit-banging driver

The XA-Gx derivative does not incorporate on-chip I²C hardware. However, I²C functionality can be achieved by software emulation. The file I2CBITS.C (Appendix II) performs two main tasks: handling complete transfers that consist of one or more messages (described above, see Figure 2), and the software emulation task. The emulation task consists of: bus monitoring and control, master sending/receiving of bytes conform to the I²C protocol.

The following macro and functions are designed for master I²C-bus control:

delay	Macro for delay loop of about 1 microsecond. Needs to be tuned (in application note done for 20MHz XA). This delay is needed to insure minimum high and low clock times on the bus. Also, the hold and setup times for START and STOP conditions are met with this macro. To optimize the speed, the software (generated by the compiler) delay is measured and included in the total delay times.
SCLHigh()	Function to release (send high) the SCL pin and wait for any clock stretching peripheral devices. At this point, if requested, the user can build in time-outs.
PutByte()	Function that sends one byte of data to a slave device. After that, it checks if slave did acknowledge.
GetByte()	Function to receive one data byte from an addressed slave. and after that it sends (no)acknowledge.
GenerateStart()	Function to generate and I²C (repeated)START condition and send slave address for a message read/write.
GenerateStop()	Function to generate an I²C STOP condition, releasing the bus. It also calls the function readyProc to signal the driver is finished, and pass the status of the transfer.

DataSheet4U.com www.DataSheet4U.com

DataSheet4U.com DataSheet4U.com

www.DataSheet4U.com

4.2 PCx8584 driver

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).

Selection of either an Intel or Motorola bus interface, achieved by detection of the first WR – CS signal sequence is outside the scope of this application note, as well as the insertion of wait states needed to meet the constraints of the XA – PCF8584 bus timing. More information about the hardware interface can be found in the Philips Semiconductors application note, AN96098: *Interfacing 68000 family peripherals to the XA*.

Bus speed

The speed of the I²C-bus is controlled by 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 the the datasheet or the Application Report of the PCx8584.

Interrupt

In this applicaiton note we assume that the interrupt output of the PCF8584 is connected to external interrupt 0 input of the XA. In the initialization function, this interrupt is enabled, its priority is set as well as the general interrupt enable flag. Furthermore, a 'soft' interrupt vector is filled to point to the right interrupt handler. This is only done for debugging purposes. In a 'real' application, this should be replaced by a ROM vector.

After completing the transmission or reception of each byte (address or data), the PIN flag in the control/status register of the PCF8584 is reset to 0. This will send an interrupt to the XA (EX0) and the interrupt service (state) handler will be called (see Figure 2).

If a transfer is started, the driver interface function returns immediately. At the end of the transfer, together with the generation of a STOP condition, the driver calls a function, passing the transfer status. A pointer to this function was given by the applicaiton at the time the transfer was applied for. It is up to the user to write this function and to determine the actions that have to be done (see for example, the function I2cReady in module I2CINTFC.C).

DataSheet4U.com www.DataSheet4U.com

DataSheet4U.com DataSheet4U.com

www.DataSheet4U.com

5. DEMO PROGRAM

The modules DEMO.C and I2CINTFC.C use either one of the drivers to implement a simple application on a Microcore 6 demo / evaluation board. They are intended as examples to show how to use the driver routines.

The Microcore 6 board contains a PCx8584 I²C-bus controller, a PCF8583 real time clock and a PCF8574 I/O expander with connections to 4 LEDs. the demo program runs the LEDs 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 (microcontrollers) 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, like the *printf()* function) and it does a retry of the transfer. However, if the maximum number of retries are reached, and exception interrupt (Trap #14) is generated to give a fatal error message.

DataSheet4U.com www.DataSheet4U.com

DataSheet4U.com DataSheet4U.com

www.DataSheet4U.com

APPENDICES

Appendix I I2CINTFC.C

```
/*****************************************************************************/
/*  Name of module : I2CINTFC.C                                              */
/*  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 1996 Philips Semiconductors B.V.                     */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/*  History:                                                                 */
/*                                                                           */
/*  96–11–25    P.H. Seerden    Initial version                              */
/*                                                                           */
/*****************************************************************************/

#include ”i2cexprt.h”
#include ”i2cdrivr.h”


extern void PrintString(code char *s);     /* to send messages out using UART     */


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;
}
```

DataSheet4U.com www.DataSheet4U.com

DataSheet4U.com DataSheet4U.com

www.DataSheet4U.com


```
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);
}
```

DataSheet4U.com www.DataSheet4U.com

DataSheet4U.com DataSheet4U.com

www.DataSheet4U.com


```
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>...<Dnum><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();
}
```

DataSheet4U.com
www.DataSheet4U.com

DataSheet4U.com
DataSheet4U.com

www.DataSheet4U.com


```
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 separated by a repeated start.
 * PROTOCOL   : <S><Slv1A><R><A><D1><A>...<Dnum1><N>
 *                      <S><Slv2A><W><A><D1><A>...<Dnum2><A><P>
 *******************************************************************************/
{
    iicTfr.mrNessages = 2;
    iicTfr.p_message  = p_iicMsg;
    p_iicMsg[0] = msg1;
    p_iicMsg[1] = msg2;

    StartTransfer();
}
```

DataSheet4U.com www.DataSheet4U.com

DataSheet4U.com DataSheet4U.com

www.DataSheet4U.com

Appendix II I2CBITS.C

```
/*****************************************************************************/
/*  Name of module : I2CBITS.C                                               */
/*  Language       : C                                                       */
/*  Name           : P.H. Seerden                                            */
/*  Description    : Driver part for XA-G3 I2C 'bit-bang' code.              */
/*                                                                           */
/*   P1.4 and P1.5 are used for SCL and SDA.                                 */
/*   Everything between one Start and Stop condition is called a TRANSFER.   */
/*   One transfer consists of one or more MESSAGEs.                          */
/*   MESSAGEs are separated by Repeated Staarts.                             */
/*   To start a transfer call function "I2C_Transfer".                       */
/*                                                                           */
/*          (C) Copyright 1996 Philips Semiconductors B.V.                   */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/*  History:                                                                 */
/*                                                                           */
/*  96-11-25    P.H. Seerden    Initial version                              */
/*                                                                           */
/*****************************************************************************/

#include <xa.h>

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


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 bit noAck;



/* about 1 us delay time at 20 MHz.                                          */
/* Used to insure minimum high and low clock times on the I2C bus.           */
/* This macro must be tuned for the actual oscillator frequency used.        */
/* Other parameters involved:   XA bus timing (BTRH and BTRL)                */
/*                              required I2C bus speed (normal or fast)      */
/*                              performance of compiler generated code       */

#define   delay    asm("  nop"); \
                   asm("  nop"); \
                   asm("  nop"); \
                   asm("  nop"); \
                   asm("  nop"); \
                   asm("  nop"); \
                   asm("  nop"); \
                   asm("  nop")
```

DataSheet4U.com
DataSheet4U.com
DataSheet4U.com
www.DataSheet4U.com

DataSheet4U.com
DataSheet4U.com

www.DataSheet4U.com


```
static void SCLHigh(void)
/************************
 * Input(s)    : none.
 * Returns     : none.
 * Description : Sends SCL pin high and wait for any clock stretching
 *               peripherals.
 ************************************************************************/
{
    SCL = 1;
    while (!SCL)  ;
    delay;
}


static void PutByte(BYTE i)
/*************************
 * Input(s)    : i       byte to be transmitted.
 * Returns     : None.
 * Description : Sends one byte of data to a slave device.
 ************************************************************************/
{
  BYTE n;

    for (n=0x80; n!=0; n=n>>1)
    {
        SDA = (i & n) ? 1 : 0;
        SCLHigh();               /* make SCL high and check for stretching */
        delay;                   /* extra delay needed                     */
        SCL = 0;
//        delay;                 /* not needed, enough delay in sw loop    */
    }
    delay;                       /* extra delay needed (1 us)              */
    SDA = 1;                     /* release data line for acknowledge      */
    SCLHigh();                   /* make SCL high and check for stretching */
    noAck = SDA;                 /* check acknowledge                      */
    SCL = 0;
}


static BYTE GetByte(void)
/************************
 * Input(s)    : None.
 * Returns     : received byte.
 * Description : Receive one byte of an addressed slave.
 ************************************************************************/
{
  BYTE n,i;

    for (n=0; n<8; n++)                     /* read 8 bits               */
    {
        SCLHigh();               /* make SCL high and check for stretching */
        i = i | SDA;
        SCL = 0;
        delay;
        i = i<<1;
    }
    SDA = noAck;
    SCLHigh();                   /* make SCL high and check for stretching */
    SCL = 0;
    SDA = 1;
    return i;
}
```

DataSheet4U.com www.DataSheet4U.com

DataSheet4U.com DataSheet4U.com

www.DataSheet4U.com


```
static void GenerateStart(void)
/*****************************
 * Input(s)    : None.
 * Returns     : None.
 * Description : Generate a start condition, and send slave address.
 *****************************************************************************/
{
    SCL = 1;                          /* needed for repeated start          */
    SDA = 1;
    noAck = FALSE;                        /* clear no ack status flag       */

    if (SCL && SDA)                       /* both lines high ??             */
    {
        SDA = 0;
        delay;
        delay;                     /* hold time start condition min. 4 us   */
        delay;
        SCL = 0;
        PutByte(msg–>address);
    }
    else
        readyProc(I2C_ERR, mssgCount);       /* Signal driver is finished  */
}


static void GenerateStop(BYTE status)
/***********************************
 * Input(s)    : status       status of the driver.
 * Returns     : None.
 * Description : Generate a stop condition, releasing the bus.
 *****************************************************************************/
{
    SDA = 0;
    SCLHigh();                   /* make SCL high and check for stretching */
    SDA = 1;                     /* stop condition setup time min. 4 us    */
    state = ST_IDLE;
    readyProc(status, mssgCount;           /* Signal driver is finished    */
}
```

DataSheet4U.com www.DataSheet4U.com

DataSheet4U.com DataSheet4U.com

www.DataSheet4U.com


```
void StateHandler(void)
/*********************
 * Input(s)    : None.
 * Returns     : None.
 * Description : Master mode state handler for I2C bus.
 ***************************************************************************/
{
    switch (state)
    {
     case ST_SENDING :
        if (noAck)
             GenerateStop(I2C_NACK_ON_DATA);
        else
            if (dataCount < msg->nrBytes)
                PutByte(msg->buf[dataCount++]);        /* sent next byte   */
            else
            {
                if (msgCount < tfr->nrMessages)
                {
                    dataCount = 0;
                    msg = tfr->p_message[mssgCount++];
                    state = (msg->address & 1) ? ST_AWAIT_ACK : ST_SENDING;
                    GenerateStart();
                }
                else
                    GenerateStop(I2C_OK);              /* transfer ready   */
            }
            break;
         case ST_AWAIT_ACK :
            if (noAck)
                GenerateStop(I2C_NACK_ON_ADDRESS);
            else
                if (msg->nrBytes == 1)
                {
                    noAck = TRUE;                      /* clear ACK        */
                    state = ST_RECV_LAST;
                }
                else
                    state = ST_RECEIVING;
            break;
         case ST_REEIVING :
          msg->buf[dataCount++] = GetByte();
            if (dataCount + 1 == msg->nrBytes)
            {
                noAck = TRUE;                         /* clear ACK         */
                state = ST_RECV_LAST;
            }
            break;
         case ST_RECV_LAST :
            msg->buf[dataCount] = GetByte();
            if (mssgCount<tfr->nrMessages)
            {
                dataCount = 0;
                msg = tfr->p_message[mssgCount++];
                state = (msg->address & 1) ? ST_AWAIT_ACK : ST_SENDING;
                GenerateStart();
            }
            else
                GenerateStop(I2C_OK);                /* transfer ready     */
            break;
         case ST_IDLE :
            break;
         default :                                /* impossible             */
            GenerateStop(I2C_ERR);                /* just to be sure        */
            break;
    }
}
```

DataSheet4U.com
www.DataSheet4U.com

DataSheet4U.com
DataSheet4U.com

www.DataSheet4U.com


```
void I2C_Transfer(I2C_TRANSFER *p, void (*proc)(BYTE, BYTE))
/*************************************************************
 * Input(s)   : p            address of I2C transfer parameter block.
 *              proc         procedure to call when trnasfer 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 = 1;
    dataCount = 0;
    msg = tfr->p_message[0];                    /* first message          */

    state = (msg->address & 10 ? ST_AWAIT_ACK : ST_SENDING;

    GenerateStart();

    while (state != ST_IDLE)
        StateHandler();
}

void I2C_Initialize(BYTE dum)
/****************************
{
    state = ST_IDLE;

    P1CFGA = P1CFGA & 0xcf;        /*P1.4 and P1.5 as open drain ports     */
    P1CFGB = P1CFGB & 0xcf;
}
```

et4U.com DataSheet4U.com DataShee DataSheet4U.com DataSheet4U.com www.DataSheet4U.com

DataSheet4U.com DataSheet4U.com

www.DataSheet4U.com

Appendix III I2C8584.C

```
/*****************************************************************************/
/*  Name of module : I2C8584.C                                               */
/*  Language       : C                                                       */
/*  Name           : P.H. Seerden                                            */
/*  Description    : Interrupt driven driver for the XA-Gx and the           */
/*                   PCx8584 I2C bus controller.                             */
/*                                                                           */
/*   Uses external interrupt 0 of the XA.                                    */
/*   Everything between one Start and Stop condition is called a TRANSFER.   */
/*   One transfer consists of the one or more MESSAGEs.                      */
/*   To start a transfer call function "I2C_Transfer".                       */
/*                                                                           */
/*        (C) Copyright 1996 Philips Semiconductors B.V.                     */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/*  History:                                                                 */
/*                                                                           */
/*  96-11-25    P.H. Seerden    Initial version                              */
/*                                                                           */
/*****************************************************************************/

#include <xa.h>

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


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

#define BYTE_AT(x)   (*(far unsigned char*)x)
                                    /*                      ES0 ES1 ES2   */
#define AR_8584    BYTE_AT(0xF0000)    /* Address Register    0   0   0    */
#define VR_8584    BYTE_AT(0xF0000)    /* Vector Register     0   0   1    */
#define CL_8584    BYTE_AT(0xF0000)    /* Clock Register      0   1   0    */
#define DR_8584    BYTE_AT(0xF0000)    /* Data Register       1   0   0    */

#define CR_8584    BYTE_AT(0xF0002)    /* Control Register    0   x   x    */
#define CS_8584    BYTE_AT(0xF0002)    /* Cntrl/Status Reg    1   x   x    */

/*****************************************************************************/


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)   : driver status to the upper layer.
 * Returns     : none.
 * Description : Generate a stop condition.
 ***************************************************************************/
{
    CR_8584 = PIN_MASK | ESO_MASK | STO_MASK | ACK_MASK;
    state   = ST_IDLE;

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


DataSheet4U.com
DataSheet4U.com
www.DataSheet4U.com

www.DataSheet4U.com


```
interrupt void I2C_Interrupt(void)
/*********************************
 * Input(s)    : none.
 * Output(s)   : none.
 * Returns     : none.
 * Description : Interrupt handler for PCF8584 int. at external int 0 pin.
 ************************************************************************/
{
    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[msgCount++];
                    state = (msg->address & 1) ? ST_AWAIT_ACK : ST_SENDING;
                    CS_8584 = ESO_MASK | STA_MASK | ACK_MASK;
                    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;                     /* clear ACK        */
                state = ST_RECV_LAST;
            }
            else
                state = ST_RECEIVING;
            dummy = DR_8584;     /* start generation of clock pulses
                                    for the first byte to read           */
        }
        break;
     case ST_RECEIVING :
        if (dataCount + 2 == msg->nrBytes)
        {
            CS_8584 = ESO_MASK;                         /* 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;
            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;
    }
}
```

DataSheet4U.com
www.DataSheet4U.com

DataSheet4U.com
DataSheet4U.com

www.DataSheet4U.com


```
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;
    msg = tfr->p_message[mssgCount++];

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


void I2C_Initialize(BYTE speed)
/*****************************
 * Input(s)   : speed      clock register value for bus speed.
 * Output(s)  : None.
 * Returns    : None.
 * Description: Initialize the PCF8584.
 ************************************************************************/
{
    state     = ST_IDLE;
    readyProc = NULL;

    AR_8584 = 0x26;                       /* dummy own slave address      */
    CR_8584 = 0x20;                       /* write clock register         */
    CL_8584 = speed;

/* for Microcore 6 and XTEND,                                             */
/* now fill the secondary vector table with the right interrupt vector    */
#asm
    mov.w  r2,#680h
    mov.w  r0,#_I2C_Interrupt&(0+65535)
    mov.w  r1,#seg _I2C_Interrupt
    mov.w  [r2+],r0
    mov.w  [r2],r1
#endasm

    CR_8584 = ESO_MASK;                  /* set serial interface ON       */

    IPA0 = IPA0 | 7;
    EX0 = 1;
    EA = 1;                             /* General interrupt enable       */
}
```

DataSheet4U.com
DataSheet4U.com
DataSheet4U.com
www.DataSheet4U.com
DataSheet4U.com
DataSheet4U.com

www.DataSheet4U.com

Appendix IV I2CDEMO.C

```
/*****************************************************************************/
/*  Name of module    : DEMO.C                                               */
/*  Program language  : C                                                    */
/*  Name              : P.H. Seerden                                         */
/*  Description       : XA-Gx I2C driver test (PCF8584 + bit bang)           */
/*                      Runs on MICROCORE 6                                  */
/*                      Read time from the real time clock chip PCF8583.     */
/*                      run leds connected to PCF8574 every second.          */
/*                                                                           */
/*        (C) Copyright 1996 Philips Semiconductors B.V.                     */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/*  History:                                                                 */
/*                                                                           */
/*  96-11-25    P.H. Seerden    Initial version                              */
/*                                                                           */
/*****************************************************************************/

#include "i2cexprt.h


#define PCF8574_WR   0x40                  /* i2c address I/O poort write */
#define PCF8574_RD   0x41                  /* i2c address I/O poort read  */
#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(0x10);       /* for PCF8584, 4.43MHz and SCL = 90KHz    */

    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);
}
```

DataSheet4U.com
www.DataSheet4U.com

DataSheet4U.com
DataSheet4U.com

www.DataSheet4U.com


```
void main(void)
{
  BYTE  oldseconds,port;

    Init();

    oldseconds = 0;
    port = 0xf7;
    while (1)
    {
        rtcBuf[0] = 2;                                  /* read seconds            */
        I2C_WriteRepRead(&rtcMsg1, &rtcMsg2);

        if (rtcBus[0] != oldseconds)                    /* one second passesed ?   */
        {
            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);
        }
    }
}
```

et4U.com DataSheet4U.com DataShee
DataSheet4U.com www.DataSheet4U.com

DataSheet4U.com
DataSheet4U.com

www.DataSheet4U.com

Appendix V I2CEXPRT.H

```
/*************************************************************************/
/*  Name of module : I2CEXPRT.H                                          */
/*  Language       : C                                                   */
/*  Name           : P.H. Seerden                                        */
/*  Description    : This module consists of 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 1996 Philips Semiconductors B.V.                 */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/*  History:                                                             */
/*                                                                       */
/*  96-11-25    P.H. Seerden    Initial version                          */
/*                                                                       */
/*************************************************************************/

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_TIME_OUT            6       /* err: Time out occurred        */


/*************************************************************************/
/*       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_Initialixe(BYTE speed);

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);
```

DataSheet4U.com
www.DataSheet4U.com

DataSheet4U.com
DataSheet4U.com

www.DataSheet4U.com

Appendix VI I2CDRIVR.H

```
/*************************************************************************/
/*  Name of module : I2CDRIVR.H                                          */
/*  Language       : C                                                   */
/*  Name           : P.H. Seerden                                        */
/*  Description    : This module contains a number of 'local'            */
/*                   declarations for the XA-Gx I2C driver package.      */
/*                                                                       */
/*        (C) Copyright 1996 Philips Semiconductors B.V.                 */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/*  History:                                                             */
/*                                                                       */
/*  96-11-25    P.H. Seerden    Initial version                          */
/*                                                                       */
/*************************************************************************/

static bit          SDA  @ 0x38C          /* port P1.4                   */
static bit          SCL  @ 0x28D;         /* port P1.5                   */


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

#define ACK_MASK            0x01
#define STO_MASK            0x02
#define STA_MASK            0x04
#define ESO_MASK            0x48            /* also interrupt enable     */

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

DataSheet4U.com
www.DataSheet4U.com

DataSheet4U.com
DataSheet4U.com

www.DataSheet4U.com

et4U.com

DataSheet4U.com

Philips Semiconductors and Philips Electronics North America Corporation reserve the right to make changes, without notice, in the products, including circuits, standard cells, and/or software, described or contained herein in order to improve design and/or performance. Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no license or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified. Applications that are described herein for any of these products are for illustrative purposes only. Philips Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

LIFE SUPPORT APPLICATIONS

Philips Semiconductors and Philips Electronics North America Corporation Products are not designed for use in life support appliances, devices, or systems where malfunction of a Philips Semiconductors and Philips Electronics North America Corporation Product can reasonably be expected to result in a personal injury. Philips Semiconductors and Philips Electronics North America Corporation customers using or selling Philips Semiconductors and Philips Electronics North America Corporation Products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors and Philips Electronics North America Corporation for any damages resulting from such improper use or sale.

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

© Copyright Philips Electronics North America Corporation 1997
All rights reserved. Printed in U.S.A.

Let's make things better.

www.DataSheet4U.com

Philips Semiconductors

PHILIPS

PHILIPS

DataSheet4U.com