MICROCHIP

AN683

How to Use the 24LCS61/62 Software Addressable Serial EEPROM

Author: Rick Stoneking
Microchip Technology Inc.

INTRODUCTION

The purpose of this application note is to provide an example of how to use the Microchip Technology 24LCS61/62 in a multi board type of system. A hypothetical system is presented and described, along with the source code to implement all of the features of the these devices.

SYSTEM DESCRIPTION

The hypothetical system described in this application note is an industrial controller that can be configured by adding/removing/changing individual circuit cards. This system utilizes a high speed parallel bus for normal communication between the individual cards, and an I^{2}C bus is used for detecting the installed system cards, determining their configuration, and initializing the system accordingly. After the initial power up sequence determines the 'boot configuration', this secondary bus is also periodically polled looking for new devices/cards.

HARDWARE DESCRIPTION

Figure 1 shows a block diagram of the system. This application note does not address the entire system, but only those portions necessary to demonstrate the code required for a typical implementation using the 24LCS61/62 devices. The system consists of a system master which controls the I^{2}C bus, and any number of additional circuit cards.

Each of these circuit cards has either a 24LCS61 or 24LCS62 on it, which allows the system master to identify the presence of each card, determine the type of card (e.g. Memory, I/O, Display), and read/write any necessary configuration data. Each circuit card may or may not have a microcontroller of it's own. For those that do the $\overline{\text{EDS}}$ pin of the 24LCS61/62 device is used as an input to the microcontroller to tell it that it has been detected by the system master.

SOFTWARE DESCRIPTION

Figure 2 shows the flow chart for the system master node initialization. At the start of the initialization the system master enters a software loop which sends the Assign ID command over the I^{2}C bus and looks for an acknowledge (ACK) signal, indicating that there is an unassigned device. If the ACK is detected the system master then reads the six byte device serial number which latches the assigned ID into the addressed 24LCS61/62 device. The serial number serves as an arbitration mechanism only in this system, nothing further is done with it after it is read. The system master then reads the first byte of the 24LCS61/62 EEPROM memory, which is defined to contain the size in bytes of the EEPROM array. This is so that the system master can write a time stamp to the last three bytes of the EEPROM after configuration has been completed. During this read the system master sets the $\overline{\text{EDS}}$ pin of the 24LCS61/62 low to signal the microcontroller (if any) on the circuit card that it has been detected and assigned.

FIGURE 1: SYSTEM BLOCK DIAGRAM

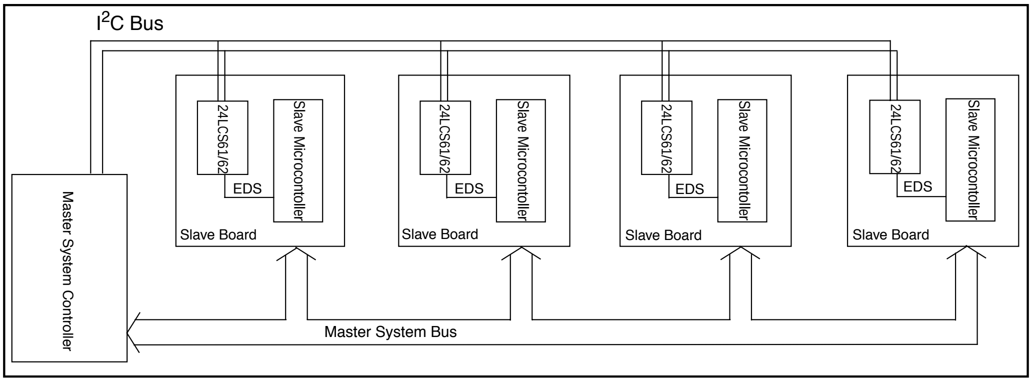
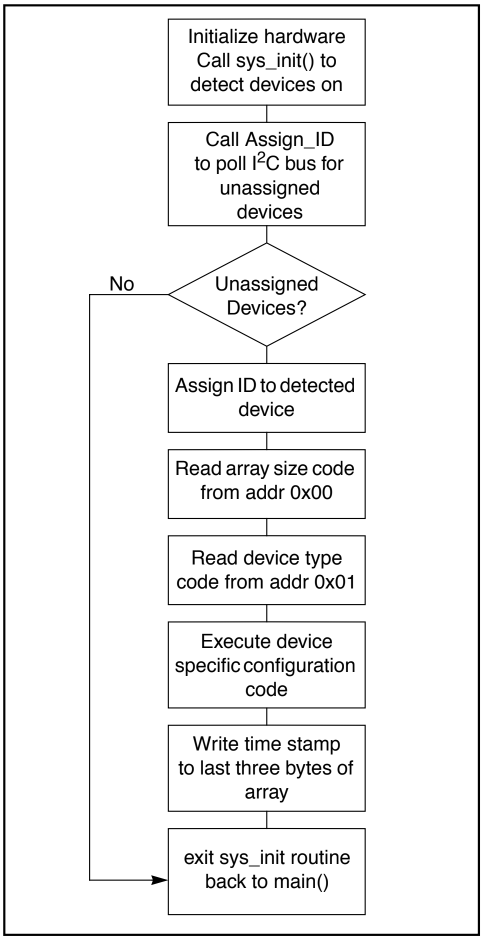


FIGURE 2: SYSTEM FLOW FOR MASTER NODE INITIALIZATION


Initialize hardware
Call sys_init() to detect devices on
Call Assign_ID to poll I2C bus for unassigned devices
Unassigned Devices?
No
Assign ID to detected device
Read array size code from addr 0x00
Read device type code from addr 0x01
Execute device specific configuration code
Write time stamp to last three bytes of array
exit sys_init routine back to main()

SOURCE CODE DISCUSSION

Appendix A contains the 'C' source code for the functions necessary to perform the operations described in the Software Description section above. These functions rely on additional I^{2}C functions that are defined in the sw12c16.lib library created by, and available from, Microchip Technology. A listing for this library has not been included in this application note.

The following files are required in order to complete the source code in Appendix A:

File Name	Description
24lcs6x.c	Main file (listing 1)
swi2c16.h	MPLAB-C header file
swi2c16.lib	MPLAB-C library file
delays.h	MPLAB-C header file
delays.lib	MPLAB-C library file
math.h	MPLAB-C header file
17c42a.h	MPLAB-C header file

The source code was compiled using MPLAB-C v1.21 and has been verified to work. The source code, including the required libraries and header files, is available for download from the Microchip website (www.microchip.com).

The following is a discussion of the purpose and operation of each of the functions in Appendix A.

Main()

This simple function represents the main system master control program. This routine sets up the system master hardware (**init_hardware()**) and then calls the **sys_init()** routine.

Sys_Init()

This routine controls the detection and configuration of any circuit cards that are installed in the system. It performs a while loop that calls the **Assign_ID()** function, passing in the id value to be assigned, to determine if there are any unassigned devices on the bus. If the **Assign_ID()** function finds a device on the bus the **sys_init()** routine then reads the memory size (address 0) and stores this in the global variable *addr* for later use. The variable *oe_bit* is then set and the device code is read (address 1). This value is used determine and execute the proper initialization/configuration code.

Note: the actual code has not been included in this example but, rather, comments are used to indicate where this code would go.

After the device configuration is finished a time stamp is written to the last three bytes of the array. Before returning to **main()** the value of the variable *id* is decremented so that a subsequent call to the **sys_init()** routine, which could be used to check for new devices that have been added to the system, will assign the next unused ID number.

© 1998 Microchip Technology Inc.

Write_Byte()

This routine writes a single byte to the 24LCS61/62 device. The address that is to be written to must be placed in the global variable *addr* before calling this function in order for it work properly. The ID number of the device to be written to, and the data byte that is to be written are passed in by the calling function. The write is then initiated by issuing the write command, followed by the device ID, address byte and data byte.

Note: The state of the global variable *oe_bit* is logically OR'd, after being shifted left three bits, with the control code in order to provide control of the state of the $\overline{\text{EDS}}$ pin.

After the data byte is sent, a stop condition is generated to initiate the internal write cycle. The **Ack_Poll()** routine is then called to wait for the device to finish writing before continuing.

Read_Byte()

This routine reads a single byte from the 24LCS61/62 device. The device ID number and the address to be read are passed in by the calling function. The routine then sets the internal address pointer of the 24LCS61/62 by sending a write command (after ORing the *oe_bit* - see **Write_Byte()** above), followed by the device ID, and the address. A start condition is then generated followed by the read command (the *oe_bit* is again OR'd), and the device ID. The data byte is then read from the I^{2}C bus and returned to the calling function after generating a stop condition.

Ack_Poll()

This is a simple subroutine that is used after a write command is sent in order to allow the 24LCS61/62 to complete an internal write cycle before continuing. This is done by issuing repeated write commands to the 24LCS61/62 device, whose ID is passed in by the calling function, until an acknowledge is generated by the 24LCS61/62 indicating that it has completed the write cycle. It should be noted that the *oe_bit* is OR'd with the write command so that the state of the $\overline{\text{EDS}}$ pin is not inadvertently changed during the polling. After the acknowledge is received a stop condition is sent before the routine ends.

Clear_Addr()

This routine issues the Clear Address command to all 24LCS61/62 devices on the bus, which causes them all to reset their internal ID to 00, and to enter the unassigned state. The Assign ID command is followed by a dummy ID byte, and a stop condition.

Assign_ID()

Checks the bus for, and assigns an ID to, an unassigned 24LCS61/62 device. To do this the Assign ID command is sent, followed by the ID (which has been passed in by the calling function). The bus is then checked for an acknowledge being generated by an unassigned device. If no ACK is seen the function returns a '0' to indicate that no device was found. If an ACK has been generated, the function then reads the six byte serial number, generates a stop condition which latches the assigned ID number into the 24LCS61/62, and returns a '1' indicating a device was found and assigned.

init_hardware()

This function initializes the hardware for the simplified system used to test and validate the operation of the code in this application note. This hardware uses a PIC17C4x with SCL and SDA on port pins RC6 and RC7 respectively. Both pins require an appropriate pullup resistor as detailed in the sw12c16.h file.

SUMMARY

This application note has detailed the basics of using the 24LCS61/62 in a multi board system. Several additions/enhancements could easily be added including implementing multi-byte page mode writes to the 24LCS61/62, detecting/handling I^{2}C bus errors, and creating the necessary code/interrupt to generate a periodic call from **main()** to **sys_init()** in order to detect and configure newly added devices. The source code provided was written in 'C", versus assembly language, for a couple of reasons. First, the use of 'C' makes the code more portable so that it can easily be implemented on any one of several microcontrollers. Second, it is believed that 'C' is a more 'readable' and therefore better serves as an instructional tool than does assembly language.

ADDITIONAL INFORMATION

24LCS61/62 Device Datasheet; Microchip Technology Inc.; DS21226

AN676: *"Physical Slot Identification Techniques for the 24LCS61/62"*; Microchip Technology, Inc.; DS00676

APPENDIX A: SOURCE CODE

```
/*****************************************************************
*                                                                *
*  24LCS6x.c                                                     *
*  Source code for application note AN683 which demonstrates     *
*  using the Microchip Technology 24LCS61/62 software            *
*  addressable I2C Serial EEPROM.                                *
*                                                                *
*  06/xx/98    Original creation   R. Stoneking                  *
*              Created and compiled with MPLAB-C V1.21           *
*                                                                *
*  This code is written to run on a PIC17C4x device with SCL     *
*  and SDA on PORTC.6 and PORTC.7 respectively.  Both of these   *
*  pins require an external pullup resistor as explained in the  *
*  swi2c16.h file.                                               *
*                                                                *
*  This code does not make any attempt run the I2C bus at a      *
*  set speed, so the code will work independant of the           *
*  oscillator frequency (at faster speeds additional delays may  *
*  be required to prevent violating the I2C timing specs.        *
*                                                                *
*****************************************************************/
#include <17c42a.h>
#include <delays.h>
#include <math.h>
#include <swi2c16.h>          /* required header file from PICmicro */
                              /* library                            */
/******************* Function Prototypes***********************/
void     sys_init(void);
void     Clear_ID(void);
char     Assign_ID(char);
void     Write_Byte(char, char);
char     Read_Byte(char, char);
void     AckPoll(char);
void     init_hardware(void);
/*********** Global Defines*************************************/
#define  SET_WP_CMD           0x60
#define  READ_CMD             0x61
#define  WRITE_CMD            0x62
#define  ASN_ADD_CMD          0x64
#define  CLR_ADD_CMD          0x66
#define  FOUND                1
#define  NFOUND               0
/*********** Global Variables**********************************/
char     temp,
         id,
         oe_bit,
         rxbuf[16],
         addr,
         hour = 5,
         minute = 0x45,
         second = 0x33;

/*****************************************************************/
void main(void)
{
         init_hardware();     /* set up PICmicro */
         Clear_ID();
         sys_init();
         while(1);            * loop forever */
}        /* end main () */
```

© 1998 Microchip Technology Inc.

```
/*****************************************************************
*  Name:   sys_init                                              *
*  Desc:   This routine controlls the querrying of the bus,      *
*          detection and assigning of new devices, and execution *
*          of any necessary configuration routines when a new    *
*          device is found.                                      *
*                                                                *
*  Inputs: None                                                  *
*                                                                *
*  Setup:  None                                                  *
*                                                                *
*  Return: Void                                                  *
*****************************************************************/
void sys_init(void)
{
     id = 1;                            /* start assigned id's at 1 */
     oe_bit = 0;                        /* EDS pin inactive */
     while(Assign_ID(id) == FOUND)
     {
     oe_bit = 1;                        /* set EDS pin low while
                                        addressing device */
     addr = Read_Byte(id,0x00);         /* read byte 0 for array size */
     temp = Read_Byte(id,0x01);         /* read byte 1 for device code */
     switch(temp)                       /* execute module dependant code */
     {
     case 0x01:                         /* Memory Module */
                                        /* add code here for Memory Module */
                                        break;

     case 0x02:                         /* I/O Module */
                                        /* add code here for I/O Module */
                                        break;

     case 0x03:                         /* Display Module */
                                        /* add code here for Display Module */
                                        break;

      default:
                                        /* add code here for unknown Module codes */
                                        break;
     }
     Write_Byte(id, second);            /* write time stamp to last 3
                                        bytes of array */
     addr--;
     Write_Byte(id, minute);
     addr--;
     oe_bit = 0;                        /* set EDS pin high during
                                        last command */
     Write_Byte(id, hour);
     id++;
     } /* end while() */
     id--;                              /* decrement id by one so next
                                        call to sys_init uses correct id */
}    /* end sys_init() */
/*****************************************************************
*  Name:   Write_Byte                                            *
*  Desc:   Write the data byte passed by the calling funtion to  *
*          the device specified in id, at the location pointed to*
*          by the global variable addr.                          *
*                                                                *
*  Inputs: id - the id of the device on the bus to be written to *
*          data_byte - byte to be written to the array           *
*                                                                *
*  Setup:  Global variable addr must be set before entry         *
*                                                                *
*  Return: Void                                                  *
*****************************************************************/
```

```
void     Write_Byte(char wr_id, char data_byte)
{
         RestartI2C();
         temp = WriteI2C(WRITE_CMD | (oe_bit << 3));
         AckI2C();
         temp = WriteI2C(wr_id);
         AckI2C();
         temp = WriteI2C(addr);              /* address byte */
         AckI2C();
         temp = WriteI2C(data_byte);         /* data byte */
         AckI2C();
         StopI2C();
         AckPoll(wr_id);                     /* wait for write cycle to finish */
}        /* end Write_Byte()                 */
/*****************************************************************
*  Name:   Read_Byte                                             *
*  Desc:   Reade the data from the device specified by id at the *
*          address specified by the paramater addr.              *
*                                                                *
*  Inputs: id - the id of the device on the bus to be written to *
*          addr - byte to be read in the array                   *
*                                                                *
*  Setup:  None                                                  *
*                                                                *
*  Return: Data byte read                                        *
*****************************************************************/
char Read_Byte(char rd_id, char loc)
{
         PORTD.0 = 1;
         StartI2C();
         temp = WriteI2C(WRITE_CMD | (oe_bit << 3));
         AckI2C();
         temp = WriteI2C(rd_id);
         AckI2C();
         temp = WriteI2C(loc);               /* address byte */
         AckI2C();
         NOP();
         RestartI2C();
         temp = WriteI2C(READ_CMD | (oe_bit << 3));
         AckI2C();
         temp = WriteI2C(rd_id);
         AckI2C();
         temp=ReadI2C();
         AckI2C();
         StopI2C();
         return I2C_BUFFER;
}        /* end Read_Byte()                  */
/*****************************************************************
*  Name:   AckPoll                                               *
*  Desc:   Send repeated Write commands and check for DUTk.      *
*          Return when ack received.  If no ack is received this *
*          will loop indefinately.                               *
*                                                                *
*  Inputs: Id of device to be polled                             *
*                                                                *
*  Setup:  None                                                  *
*                                                                *
*  Return: Void                                                  *
*****************************************************************/
void     AckPoll(char ack_id)
{
         do
    {
         RestartI2C();
         temp = WriteI2C(WRITE_CMD | (oe_bit << 3));
         AckI2C();
         temp = WriteI2C(ack_id);
         AckI2C();
```

© 1998 Microchip Technology Inc.

```
     }while(BUS_STATUS.2);
     StopI2C();
}    /* end AckPoll()           */
/****************************************************************
*  Name:   Clear_Addr()                                         *
*  Desc:   Send Clear Address Command to all devices on the bus, *
*          resetting all device ID registers to 00.             *
*                                                               *
*  Inputs: None                                                 *
*                                                               *
*  Setup:  None                                                 *
*                                                               *
*  Return: Void                                                 *
****************************************************************/
void  Clear_ID()
{
     RestartI2C();
     temp = WriteI2C(CLR_ADD_CMD);
     AckI2C();
     temp = WriteI2C(0x00);
     AckI2C();
     StopI2C();
}
/****************************************************************
*  Name:   Assign_ID                                            *
*  Desc:   Sends Assign ID command over the I2C bus and looks for*
*          the acknowlege from an unassigned device.  If a device*
*          is found, the six byte serial number is read into    *
*          the rxbuf[] array for use by the calling function.   *
*                                                               *
*  Inputs: Id to be assigned to device if detected              *
*                                                               *
*  Setup:  None                                                 *
*                                                               *
*  Return: 1 if device is found, 0 if no device found           *
****************************************************************/
char  Assign_ID(char new_id)
{
     RestartI2C();
     temp = WriteI2C(ASN_ADD_CMD);
     AckI2C();
     temp = WriteI2C(new_id);
     AckI2C();
     if (BUS_STATUS.2)                    /* Check for unassigned device on bus */
     return NFOUND;                        /* and exit if none */
     temp = getsI2C(rxbuf,6);             /* Otherwise read SN */
     StopI2C();
     return FOUND;
}
void  init_hardware()
     {
     SPBRG = 0x0f;                        /* Use 0x07 for 19.2Kbps, 0x0f for 9600 bps  */
     TXSTA = 0x20;                        /* Async mode, 8 Data bits, txen = 1         */
     RCSTA = 0x90;                        /* 8 Data bits, spen = 1, cren = 1           */
     PORTB = 0x0;
     DDRB = 0xff;                         /* Port B all inputs                         */
     PORTC = 0xff;                        /* Set Port C to all ones                    */
     DDRC = 0xff;                         /* Port C all outputs for now                */
     PORTD = 0;
     DDRD = 0x0;                           /* Port D all outputs                        */
     DDRE = 0xff;                         /* Port E all inputs                         */
     PIR = 0;                             /* Clear any pending interrupts              */
     CPUSTA.GLINTD = 1;                   /* Disable global interrupts                 */
     }                                    /*  end init_hardware()                      */

#include <swi2c16.lib>
#include <delays.lib>
```

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-786-7200 Fax: 480-786-7277
Technical Support: 480-786-7627
Web Address: http://www.microchip.com

Atlanta
Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston
Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508-480-9990 Fax: 508-480-8575

Chicago
Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
Microchip Technology Inc.
4570 Westgrove Drive, Suite 160
Addison, TX 75248
Tel: 972-818-7423 Fax: 972-818-2924

Dayton
Microchip Technology Inc.
Two Prestige Place, Suite 150
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Detroit
Microchip Technology Inc.
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Los Angeles
Microchip Technology Inc.
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York
Microchip Technology Inc.
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

AMERICAS (continued)

Toronto
Microchip Technology Inc.
5925 Airport Road, Suite 200
Mississauga, Ontario L4V 1W1, Canada
Tel: 905-405-6279 Fax: 905-405-6253

ASIA/PACIFIC

Hong Kong
Microchip Asia Pacific
Unit 2101, Tower 2
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2-401-1200 Fax: 852-2-401-3431

Beijing
Microchip Technology, Beijing
Unit 915, 6 Chaoyangmen Bei Dajie
Dong Erhuan Road, Dongcheng District
New China Hong Kong Manhattan Building
Beijing 100027 PRC
Tel: 86-10-85282100 Fax: 86-10-85282104

India
Microchip Technology Inc.
India Liaison Office
No. 6, Legacy, Convent Road
Bangalore 560 025, India
Tel: 91-80-229-0061 Fax: 91-80-229-0062

Japan
Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa 222-0033 Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Shanghai
Microchip Technology
RM 406 Shanghai Golden Bridge Bldg.
2077 Yan'an Road West, Hong Qiao District
Shanghai, PRC 200335
Tel: 86-21-6275-5700 Fax: 86 21-6275-5060

ASIA/PACIFIC (continued)

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan, R.O.C
Microchip Technology Taiwan
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

United Kingdom
Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5858 Fax: 44-118 921-5835

Denmark
Microchip Technology Denmark ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Arizona Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 München, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

11/15/99

DNV Certification, Inc.
USA

DNV MSC
The Netherlands
Accredited by the RvA

ANSI • RAB QMS ACCREDITED

DNV

ISO 9001 / QS-9000
REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.

All rights reserved. © 1999 Microchip Technology Incorporated. Printed in the USA. 11/99 Printed on recycled paper.

Information contained in this publication regarding device applications and the like is intended for suggestion only and may be superseded by updates. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights. The Microchip logo and name are registered trademarks of Microchip Technology Inc. in the U.S.A. and other countries. All rights reserved. All other trademarks mentioned herein are the property of their respective companies.