Microcontrollers

ApNote AP2427

☑ additional file
AP242701.ZIP available

Using an I²C LCD with the Infineon C500 microcontroller family. Reference code and applications including visual effects.

This application note describes the usage of I²C interface Liquid Crystal Displays with Infineon microcontrollers and presents a standard library of functions used to manipulate the Display contents efficiently. The last part includes some examples of complex visual effects. The code provided is in C and does not use any hardware-specific instructions, so it can be easily ported to Infineon 16bit microcontrollers. By replacing the I²C bus interface code with another interface code, one can use the standard library for other configurations (i.e. parallel interface) too.

Authors: Dimitrios Skraparlis / Andreas Jansen

1999-06, Rel. 01

Edition 1999-06

Published by
Infineon Technologies AG
81726 München, Germany

LEGAL DISCLAIMER

THE INFORMATION GIVEN IN THIS APPLICATION NOTE IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

Contents

AP2427 ApNote – Revision History		
Actual Release : Rel.01	Previous Release: none	
Page of Actual Rel.	Page of prev. Rel.	Subjects changes since last release)
-	-	No changes

1 Introduction

This application note focuses on Liquid Crystal Screens driven by a controller/driver chip, as shown on figure 1. This controller is programmed by the C500 Infineon microcontroller. The LCD used in this application note is shown in figure 2 and has a two pin interface to the I²C bus.

1.1 Brief introduction to the I²C bus

The I²C bus is a bidirectional two line bus, enabling communication between any kind of integrated circuit that supports this protocol, either by hardware or software. Examples of such ICs are LCD controllers, EEPROMs, RAMs, data converters or general purpose microcontrollers. The main advantage of this protocol is its two line interface, as shown in figure 3.

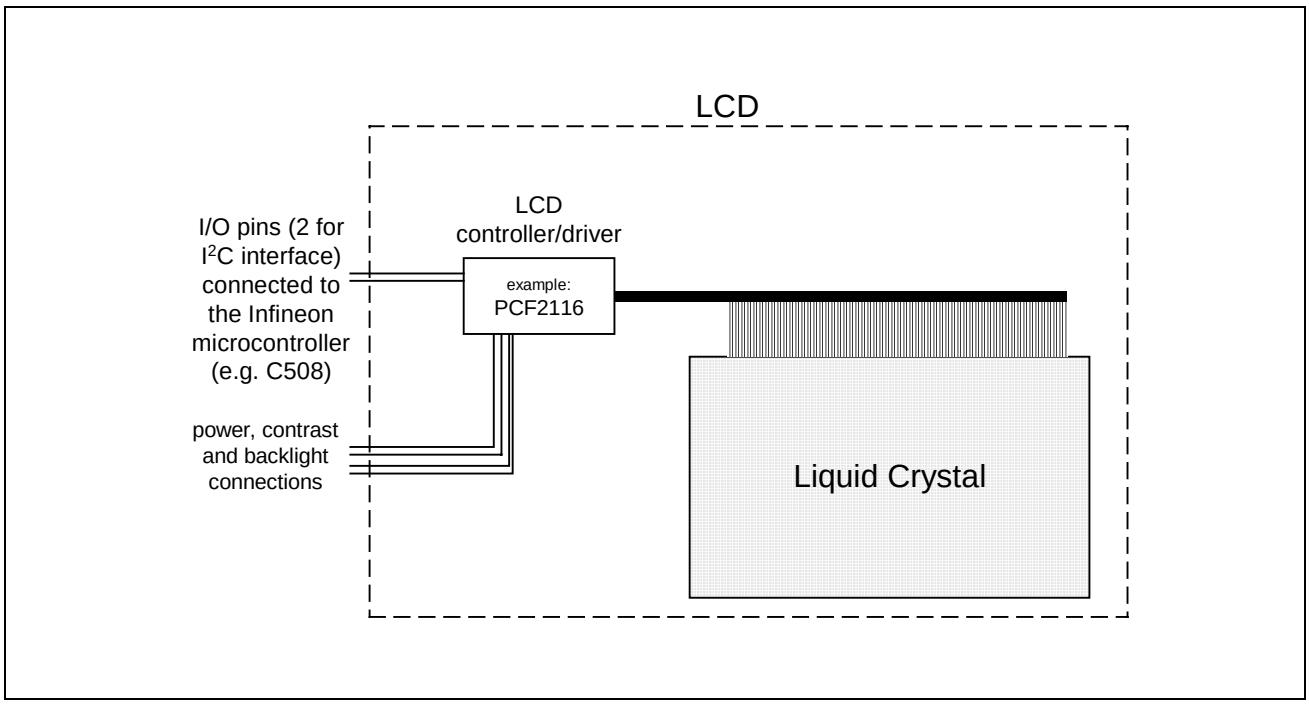


Figure 1:
Example of an LCD with an embedded controller/driver.

Figure 2:
Example of a Chip-on-glass application (3 line x 12 visible columns, I²C interface).

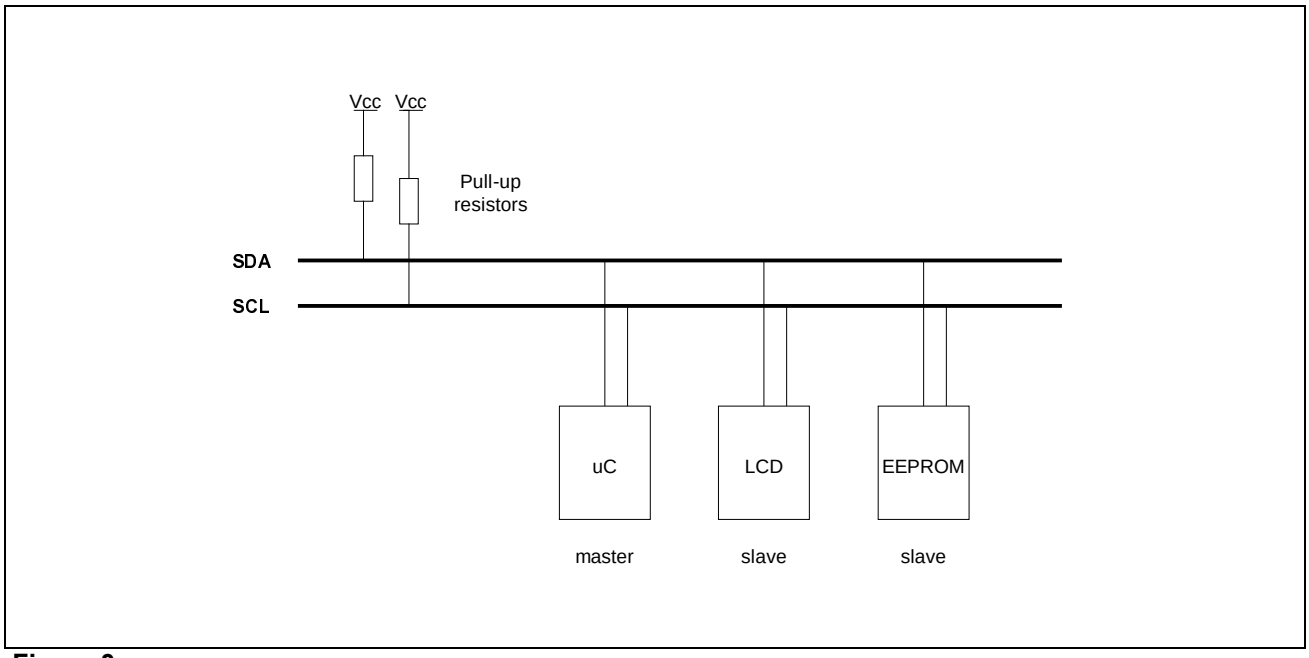


Figure 3:
Example of an I²C bus configuration.

The two-line bus consists of lines SDA (serial data line) and SCL (serial clock line). These lines should be connected to a positive supply via pull-up resistors. However this is not always necessary because some microcontroller ports (i.e. the Infineon microcontroller C504 port 1) have an internal pullup. In any case, careful consideration must be given to the I²C bus specification.

It should be noted that the multi-master capability of the I²C bus is not used in this application; there is only one master (the Infineon microcontroller) and a slave (the Display).

1.2 The communication between the master and the slave

By using the word „master“ one refers to the device which initiates and terminates a transfer and also provides the clock signals on line SCL. Master devices operate as transmitters or receivers.

At the start of each transfer, a slave is addressed by its own unique address. A transfer consists of a *start* condition, the *data bits*, an *acknowledge* bit and possibly a *stop* condition. This concept is shown in figures 4 and 5.

A *start* condition is defined by a falling edge at SDA while the SCL is high. A *stop* condition is defined by a rising edge at SDA while the SCL line is high. When transmitting data, no changes at the SDA line while the clock is high are permitted, otherwise this will result in a stop or a start condition! In order to avoid this, the master should change the data at SDA only when the SCL line is low.

After the transmission of the 8 data bits, the master sets the SDA line to high and the slave acknowledges the transfer by pulling the SDA line to ground. This indicates a successful transfer.

Master-read mode is not exactly the same. The master still provides the clock but the slave is now the one who submits the data (requested by the master) at the SDA line. At the end of the transmission, the master does not acknowledge (SDA is set and remains high).

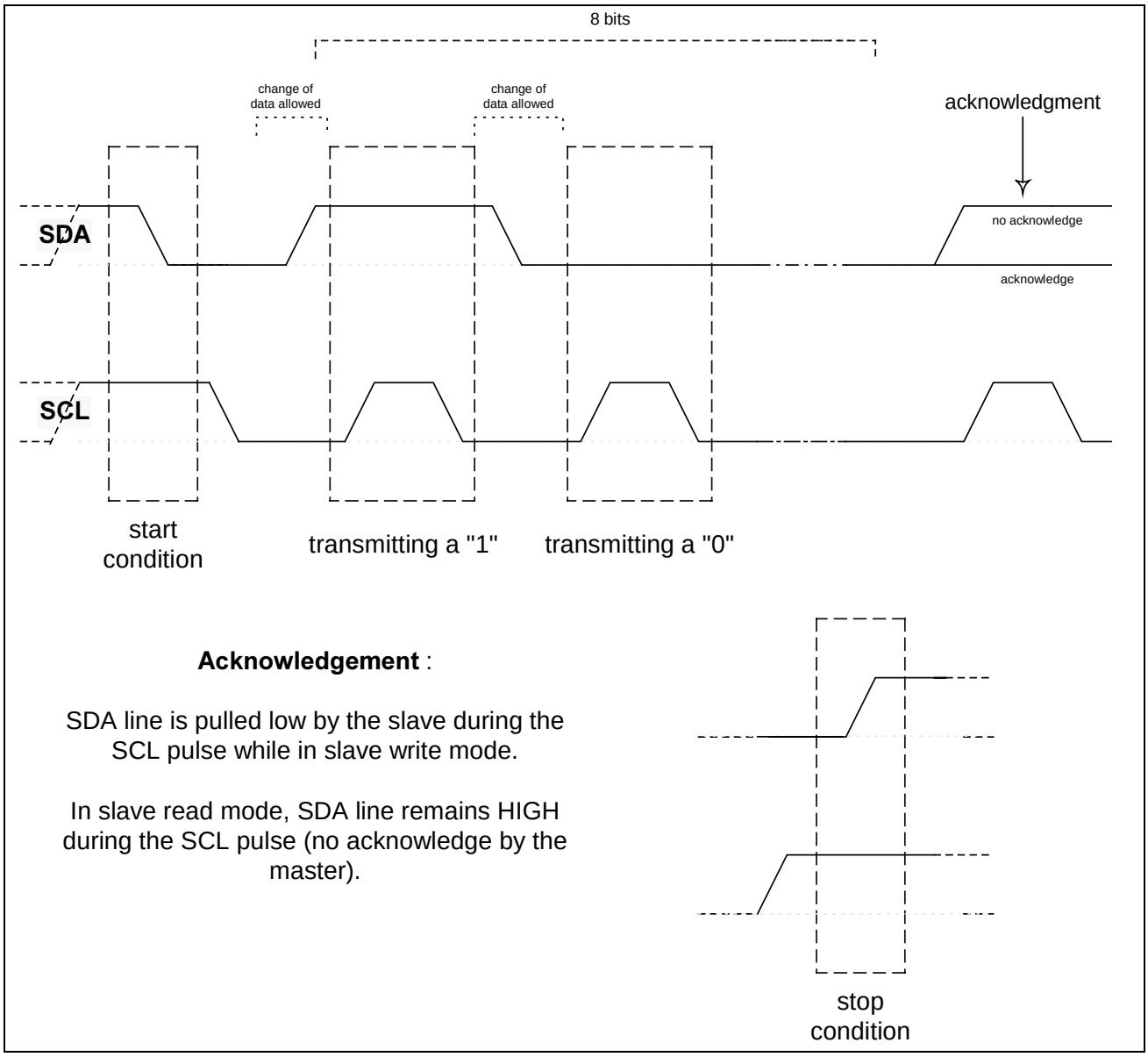


Figure 4:
An example of a transfer.

The I²C protocol defines two modes: standard mode (maximum transfer rate of 100 kbit/s) and fast mode (maximum transfer rate of 400 kbit/s). This application note uses the standard mode for communication. This could probably be easily changed but any changes should be made in accordance to the original I²C-bus specification.

For further information the interested reader is referred to the original I²C-bus specification («the I²C-bus and how to use it (including specification)», Philips Semiconductors).

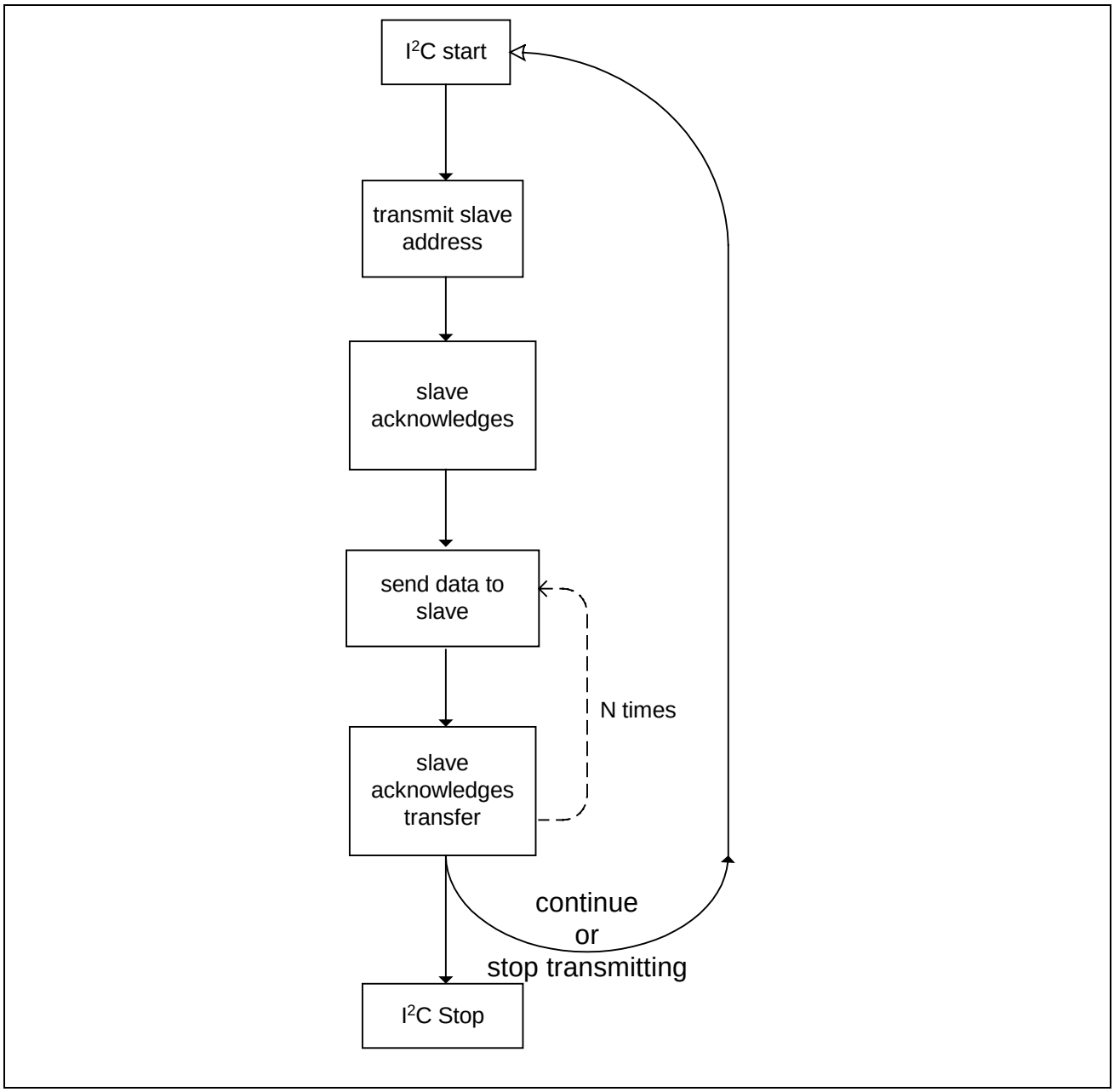


Figure 5:
An example of a successful master-write.

1.3 How is I²C implemented?

The code found in the appendix uses true software routines: the I²C is emulated by the software routines and therefore there is no need for special hardware. Nevertheless, some Infineon microcontrollers provide a hardware I²C interface. Please refer to the following table, table 1. It should be noted that software emulation is possible with any Infineon microcontrollers – one may only replace the I²C driver software with the appropriate I²C driver code for the microcontroller used. The application notes proposed can be found on the internet at www.infineon.com.

	8 bit	16 bit
software emulation	*	**
hardware implementation	-	C161RI C161PI C161SI/CI C161CS

---

* software emulated I²C interface with the C500 family. Complete description and source code is found in:

Interfacing SLx 24Cxx I2C-Bus

Serial EEPROMs to 8051 Controller Family, especially to the Siemens (Infineon's) C500 Controller Family.

AP083701

This is a source for the I²C bus procedures (e.g. I2cMasterWrite(), I2cMasterRead();) used in this application note.

** software emulated I²C interface with the C166 family. Complete description and source code is found in:

Software emulation of the I2C-bus using the General Purpose Timer unit 1 of the C166 family

This is a software emulation of the I2C-bus by using two general purpose timers of a microcontroller from the C166 family. The I2C-bus is used in many applications mainly to communicate between devices connected to the bus.

AP162401

Software emulation of I2C-bus using CPU time of C166 family

This is a software emulation of I2C-bus using CPU time of the C166 family to generate the clock and data.

AP162501

Software emulation of the I2C-bus using the High-Speed Synchronous Serial Interface of C166 family

This is a software emulation of I2C-bus using High-Speed Synchronous Serial Interface of the C166 family to generate the clock and data.

AP162601

Some versions of the C161 have hardware I²C interface capability. A related application note is:

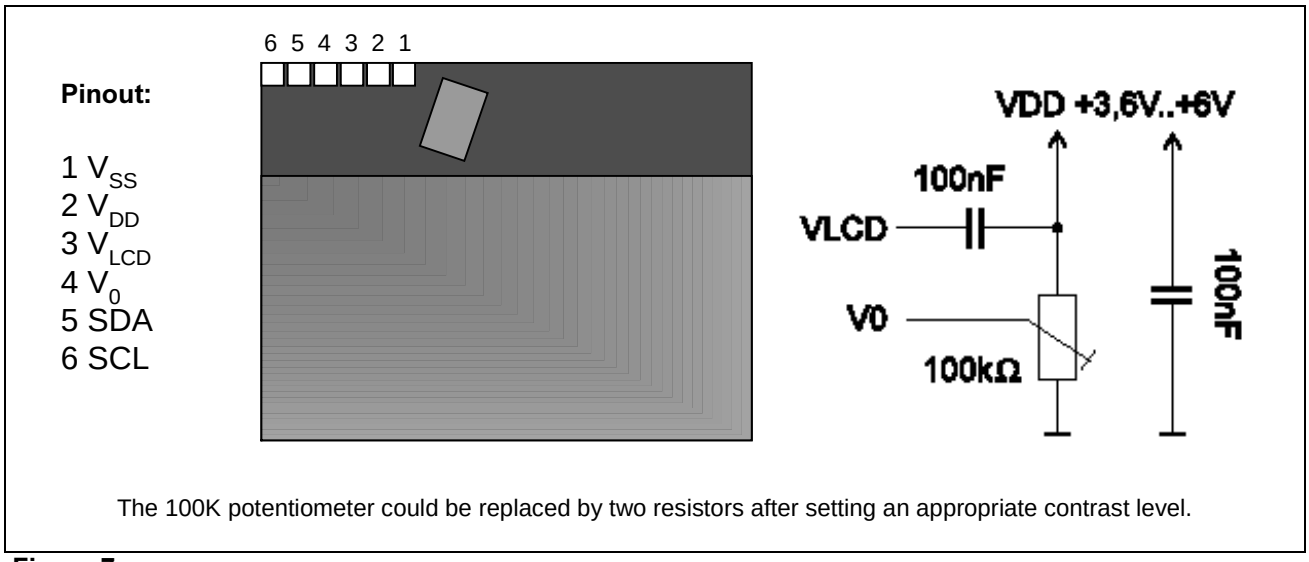
Hardware I2C-bus in slave mode by using polling and interrupt methods for C161RI microcontroller

This is a software module for hardware I2C-bus in slave mode by using software polling and hardware interrupt methods for C161RI microcontroller. The I2C-bus is used in many applications mainly to communicate between devices connected to the bus.

AP164510

1.4 The Liquid Crystal Display

The Liquid Crystal Display used in this application note is a „Chip on glass“ LCD using an I²C interface (figure 2)*. The Display controller driving the display is a PCF2116 from Philips Semiconductors. The user has access to six pins; two pins are used for I²C communication with the master - Infineon microcontroller while the other four pins are used for the power supply interfacing and for connecting 2 fixed resistors (or a potentiometer) that adjust the contrast of the Display, according to figure 5.



The 100K potentiometer could be replaced by two resistors after setting an appropriate contrast level.

Figure 7:
The LCD pinout and the contrast circuit using a single power supply (if PCF2116 bit G=1).

1.5 The LCD controller/driver

Bytes sent to the PCF2116 are either control bytes, commands or data.

A control byte selects the desired instruction group according to table 1.

- A control byte consists of 3 bits, followed by 5 zeros:

  Co RS RW‘ 0 0 0 0 0

  Bit Co=0 defines whether the following bytes are not control bytes and bit Co=1 defines that the following two bytes are a command/data byte and another control byte.

  Please note that in this application note,

  instructions with RS=0, RW‘=0 will be called „first instruction group“ instructions,

  instructions with RS=1, RW‘=0 will be called „second instruction group“ instructions and

  Other instructions are read instructions and do not abide to any instruction group; they are also not used in this application note.

  This convention will be used in the code.

- A command consists of 8 bits:

  DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0

* The ordering number is EA 7123-I2C and is distributed by Electronic Assembly: www.lcd-module.de.

A complete instruction is therefore 2 bytes:

Co RS RW‘ 0 0 0 0 0 DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0

although one could send a control byte only once (with Co=0) and then send commands and always refer to the same instruction group.

The PCF2116 instruction set is presented in the following table:

INSTRUCTION	RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	DESCRIPTION
NOP	0	0	0	0	0	0	0	0	0	0	No operation.
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in Address Counter.
Return Home	0	0	0	0	0	0	0	0	1	0	Sets DDRAM address 0 in Address Counter. Also returns shifted display to original position. DDRAM contents remain unchanged.
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies shift of display. These operations are performed during data write and read.
Display control	0	0	0	0	0	0	1	D	C	B	Sets entire display on/off (D), cursor on/off (C) and blink of cursor position character (B).
Cursor/display shift	0	0	0	0	0	1	S/C	R/L	0	0	Moves cursor and shifts display without changing DDRAM contents.
Function set	0	0	0	0	1	DL	N	M	G	0	Sets interface data length (DL), number of display lines (N, M) and voltage generator control (G).
Set CGRAM address	0	0	0	1	A_{CG}						Sets CGRAM address.
Set DDRAM address	0	0	1	A_{DD}							Sets DDRAM address.
Read busy flag and address	0	1	BF	A_C							Reads Busy Flag (BF) indicating internal operation is being performed and reads Address Counter contents.
Read data	1	1	read data								Reads data from CGRAM or DDRAM.
Write data	1	0	write data								Writes data to CGRAM or DDRAM.

Table 1a:
The PCF2116 family instruction set.

BIT	0	1
I/D	decrement	increment
S	display freeze	display shift
D	display off	display on
C	cursor off	cursor on
B	character at cursor position does not blink	character at cursor position blinks
S/C	cursor move	display shift
R/L	left shift	right shift
DL	4 bits	8 bits
G	voltage generator: $V_{LCD} = V_0$	voltage generator; $V_{LCD} = V_0 - 0.8V_{DD}$
N, (M = 0)		
PCF2116x	1 line × 24 characters; MUX 1 : 16	2 lines × 24 characters; MUX 1 : 32
PCF2114x	2 line × 12 characters; MUX 1 : 16	2 lines × 24 characters; MUX 1 : 32
N, (M = 1)	reserved	4 lines × 12 characters; MUX 1 : 32
BF	end of internal operation	internal operation in progress
Co	last control byte, only data bytes to follow	next two bytes are a data byte and another control byte

Table 1b:
Description of the command bits.

The architecture and instructions of the PCF2116 will not be fully described here. Please refer to the „PCF2116 family LCD controller/drivers“ product specification.

1.6 Using other instruction sets

It is of course possible to use other Displays, which have a controller with different commands or interface. In general, as it can be seen in table 2 (the instruction set for the Hitachi HD44710 microcontroller), the instructions are similar to the PCF2116 – that is, the manufacturers preserve a common instruction set. This is why it is very easy to program different LCDs by using the code in this application note as a start. It should be noted that it is also possible to switch to other LCD interfaces (serial or parallel) very simply because the code presented here is general (using procedures and include files). Therefore one can build his/her own interface routines and not use the I²C routines provided here. The application can then take advantage of the ready procedures in this paper, but not the optimized ones, as they are written specifically for an I²C LCD interface.

Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.
Write data to CG or DDRAM	1	0	Write data								Writes data into DDRAM or CGRAM.
Read data from CG or DDRAM	1	1	Read data								Reads data from DDRAM or CGRAM.

Table 2a:
The HD44710 instruction set.

I/D = 1: Increment I/D = 0: Decrement S = 1: Accompanies display shift S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right R/L = 0: Shift to the left DL = 1: 8 bits, DL = 0: 4 bits N = 1: 2 lines, N = 0: 1 line F = 1: 5 × 10 dots, F = 0: 5 × 8 dots BF = 1: Internally operating BF = 0: Instructions acceptable	DDRAM: Display data RAM CGRAM: Character generator RAM ACG: CGRAM address ADD: DDRAM address (corresponds to cursor address) AC: Address counter used for both DD and CGRAM addresses

Table 2b:
Description of the command bits.

1.7 Starting up – testing the LCD

Supposing that all connections are made correctly and ensuring that the I²C bus is configured properly (having appropriate pullups), one is ready to test the functionality of the LCD. A simple „HELLO!“ algorithm is appropriate. The following code is based in the example from the PCF2116 datasheet. Step 4 should be changed according to the LCD used (refer to the „function set“ instruction in the datasheet).

```
/*****************************************************************************************/
/* Test LCD                                                                              */
/*                                                                                       */
/* Description:                                                                          */
/* Send instructions to LCD according to the example in the PCF2116 datasheet.           */
/*****************************************************************************************/
#include “i2c_sw8b.c“                                          // Siemens I2C-bus module
// Some characters from character set C
#define character_space 0x20
#define character_H     0x0C8
#define character_E     0x0C5
#define character_L     0x0CC
#define character_O     0x0CF
#define character_excl  0x0A1

void main(void)
{
     I2cInit();                              // Initialize I2C bus
     I2cStart();                             // step 1 : Start communication
     I2cMasterWrite(0x74);   //01110100      // step 2 : Send SLAVE ADDRESS
     I2cMasterWrite(0x00);   //00000000      // step 3 : CONTROL BYTE (first instruction group)
     I2cMasterWrite(0x3E);   //00111110      // step 4 : _FUNCTION SET  // * change this
     I2cMasterWrite(0x0E);   //00001110      // step 5 : _DISPLAY ON/OFF CONTROL
     I2cMasterWrite(0x06);   //00000110      // step 6 : _ENTRY MODE SET
     // I2cStop(); // This is not needed

     I2cStart();                             // step 7
     I2cMasterWrite(0x74);   //01110100      // step 8 : Send SLAVE ADDRESS
     I2cMasterWrite(0x40);   //01000000      // step 9 : CONTROL BYTE (second instruction group)
     I2cMasterWrite(character_H);            // Write „H“ to DDRAM (also on the Display)
     I2cMasterWrite(character_E);            // Write „E“
     I2cMasterWrite(character_L);            // Write „L“
     I2cMasterWrite(character_L);            // Write „L“
     I2cMasterWrite(character_O);            // Write „O“
     I2cMasterWrite(character_excl);         // Write „!“
     I2cStop();                              // (step 17) // stop communication
}
```

2 The software library

The basic procedures were built arround the PCF2116 instruction set. These are:

PROCEDURE NAME	INPUT	OUTPUT of *non-optimized* procedure*			INSTRUCTION GROUP
		success	error	other	
cls_LCD	-	0	1	-	first
returnHome_LCD	-	0	1	-	
entry_mode_set_LCD	bit ID, bit S	0	1	-	
display_control_LCD	bit D, bit C, bit Blink	0	1	-	
cursor_display_shift_LCD	bit SC, bit RL	0	1	-	
function_set_LCD	-	0	1	-	
set_CGRAM_address_LCD	unsigned char address	0	1	-	
set_DDRAM_address_LCD	unsigned char address	0	1	-	
write_char2LCD	unsigned char character, unsigned char offset	0	1	-	second

Table 3:
Basic procedures that correspond to the Display Controller's instructions.

Each procedure can be called **only** after calling the appropriate procedure for the instruction group in which it belongs (otherwise the instructions will not be executed!):

*void **first_instruction_group**(void)*

*void **second_instruction_group**(void)*

Based on these procedures, one can initialize and use the LCD. To facilitate programming, certain routines were developped and are presented in table 4.

* the optimized procedures do not have output (void proc(..))

PROCEDURE NAME	INPUT	OUTPUT of *non-optimized* procedure*		
		success	error	other
wait / wait2	time_out : [0..65535]	-		
gotoXY_LCD	unsigned char pos	0	1	-
initialize_LCD	bit ID,bit S,bit D,bit C,bit Blink,bit SC,bit RL	0	1	-
writedata2LCD_reverse	unsigned char number, unsigned char *value, unsigned char offset	0	character position at which the error occured (1..number)	
writedata2LCD_normal	unsigned char number, unsigned char *value, unsigned char offset	0	character position at which the error occured (1..number)	
send_instruction2LCD	bit Co,bit RS,bit RW	0	1	-

Table 4:
Standard procedures.

2.1 Description of the basic procedures

	non-optimized:	*optimized:*
cls_LCD	unsigned char cls_LCD(void)	void cls_LCD(void)
first instruction group	**inputs:** - **outputs:** 0 if success; 1 if there was an error (could not send instruction to the LCD)	-

description:

Clears the whole display (including non-visible areas) and returns cursor to position 1. Display shift is also cancelled.

comments:

After executing this command, the master-microcontroller will wait for about 2 ms until the LCD is ready to accept new commands.

see also:

wait()

* the optimized procedures do not have output (void proc(..))

	non-optimized:	*optimized:*
returnHome_LCD	unsigned char returnHome_LCD(void)	void returnHome_LCD(void)
first instruction group	**inputs:** - **outputs:** 0 if success 1 if there was an error (could not send instruction to the LCD) **description:** Returns cursor to position 1. **comments:** - **see also:** *cls_LCD(), gotoXY_LCD()*	-

	non-optimized:	*optimized:*
entry_mode_set_LCD	unsigned char entry_mode_set_LCD(bit ID, bit S)	void entry_mode_set_LCD(bit ID, bit S)
first instruction group	**inputs:** bits ID, S. They are described in the PCF2116 datasheet **outputs:** 0 if success 1 if there was an error (could not send instruction to the LCD) **description:** Instructs how the display should behave when writing characters or when shifting the display with the cursor_display_shift_LCD command. **comments:** - **see also:** *cursor_display_shift_LCD()*	-

	non-optimized:	*optimized:*
display_control_LCD	unsigned char display_control_LCD(bit D, bit C, bit Blink)	void display_control_LCD(bit D, bit C, bit Blink)

first instruction group

inputs:

bits D, C, Blink. They are described in the PCF2116 datasheet

outputs:

0 if success 1 if there was an error (could not send instruction to the LCD)	-

description:

Sets display, cursor, or cursor-blink on or off.

comments:

-

see also:

-

	non-optimized:	*optimized:*
cursor_display_shift_LCD	unsigned char cursor_display_shift_LCD(bit SC, bit RL)	void cursor_display_shift_LCD(bit SC, bit RL)

first instruction group

inputs:

bits SC, RL. They are described in the PCF2116 datasheet

outputs:

0 if success 1 if there was an error (could not send instruction to the LCD)	-

description:

Moves the cursor or shifts the visible area of the display left or right, according to the bits ID and S (set by the entry_mode_set_LCD instruction).

comments:

When the cursor is at the end of a line and is instructed to proceed right (/ left, at the start of a line), the cursor will move to the start of the next line (/ the end of the previous line). However, shifting the display outside the line boundaries will result in a wrap-around. Cursor wrap-around occurs only between the end of the last and the start of the first line.

see also:

entry_mode_set_LCD()

	non-optimized:	*optimized:*
function_set_LCD	unsigned char function_set_LCD(void)	void function_set_LCD(void)

first instruction group

inputs:

-

outputs:

0 if success 1 if there was an error (could not send instruction to the LCD)	-

description:

Sets the bits DL,N,M,G, which are defined as constant at the start of the code, according to the type of the LCD.

comments:

DL is set to 1 when an 8 bit data transmission during any I2C transfer is desired. DL can be set to 0 when a 4 bit data transmission during any I²C transfer is desired, but *this has not been tested*.

Bits N ,M are set according to the number of lines and columns in our display. To change this, define USE_4_LINES or USE_2_LINES or USE_1_LINE at the start of the lcd.h module; one does not need to pay attention to these bits (N, M) because they are automatically properly when selecting the appropriate USE_x_LINEs definition.

Bit G should be set to one if one wishes to use a single and positive power supply. Otherwise (bit G=0 – this is default at LCD startup) one would need to use two power supplies, one negative and one positive. Please refer to the comments of the lcd.h file for the schematics.

see also:

-

	non-optimized:	*optimized:*
set_CGRAM_address_LCD	unsigned char set_CGRAM_address_LCD(unsigned char address)	void set_CGRAM_address_LCD(unsigned char address)

first instruction group

inputs:

address : [0..63] (other inputs are processed with their 5 least significant bits)

outputs:

0 if success 1 if there was an error (could not send instruction to the LCD)	-

description:

Sets the CGRAM address.

comments:

This instruction is used to define new characters/fonts. It is executed before writting to the CGRAM with the write_char2LCD() or the writedata2LCD_normal() procedures. It is also used inside the create_font() procedure.

Each line of a character is a byte (with only the 6 least significant bits used). Each bit indicates whether a dot is present or not (1 or 0). For example, the first character of the CGRAM is addresses 0000000 to 0000111 (8 lines) with each address containing a 6 bit value.

An example is found at the visual effects chapter, where an «Infineon» logo is created and shown in a complex way on screen.

see also:

create_font()

	non-optimized:	*optimized:*
set_DDRAM_address_LCD	unsigned char set_DDRAM_address_LCD(unsigned char address)	void set_DDRAM_address_LCD(unsigned char address)
first instruction group	**inputs:** address : [0..255]	
	outputs:	
	0 if success 1 if there was an error (could not send instruction to the LCD)	-
	description: Sets the DDRAM address.	
	comments: DDRAM address corresponds to the cursor position.	
	see also: -	

	non-optimized:	*optimized:*
write_char2LCD	unsigned char write_char2LCD(unsigned char character, unsigned char offset)	void write_char2LCD(unsigned char character, unsigned char offset)

second instruction group

inputs:

the desired byte to write.

the offset is added to the character number.

outputs:

0 if success 1 if there was an error (could not send instruction to the LCD)	-

description:

Writes a byte to the LCD.

comments:

when writing to the display, one must consult the appropriate character set table for the LCD used. The offset byte is used in order to convert between ASCII and the LCD character sets. For example, the ASCII «A» should be written with an offset of 0x81, in order to be displayed correctly.

see also:

writedata2LCD_normal(), writedata2LCD_reverse()

2.2 Description of the standard procedures

non-optimized & optimized:

wait	**wait2**	void wait(time_out)	void wait2(time_out)
		inputs: the desired time out **outputs:** - **description:** wait() slows the uC down for about 10*time_out clock cycles. wait2() uses a double loop (calling wait()), so one can achieve greater time-outs of several seconds. **comments:** These routines are used to create time-outs. They are very useful when creating visual effects, because they slow a process down, so that the human eye can perceive it. wait() is normally used after a cls_LCD() instruction. Here, it is included inside the cls_LCD() procedure. **see also:** *cls_LCD(), any visual effect*	

	non-optimized:	*optimized:*
gotoXY_LCD	unsigned char gotoXY _LCD(unsigned char pos)	void gotoXY _LCD(unsigned char pos)
	inputs: desired position of the cursor **outputs:** 0 if success 1 if there was an error (could not send instruction to the LCD)	-
	description: The cursor is placed in pos. **comments:** This moves the cursor to a desired position. Example: gotoXY(line1_start+3); //move to the 3+1 = 4th column, 1st line line1_start is automatically defined when selecting an LCD with the USE_x_LINEs definition **see also:** *returnHome_LCD()*	

non-optimized & optimized:

initialize_LCD	unsigned char initialize_LCD(bit ID, bit S, bit D, bit C, bit Blink, bit SC, bit RL)	void initialize_LCD(bit ID, bit S, bit D, bit C, bit Blink, bit SC, bit RL)
	inputs: bits ID, S, D, C, Blink, SC, RL **outputs:** 0 if success 1 if there was an error (could not send an instruction to the LCD) **description:** Initializes the LCD. This procedure calls all the initialization procedures. It should be included in a new project only if it is called in two or more discrete positions of the code, otherwise simple initialization-procedure calls will produce smaller code size. **comments:** - **see also:** *entry_mode_set_LCD(), display_control_LCD(), cursor_display_shift_LCD(), function_set_LCD()*	-

non-optimized: *optimized:*

writedata2LCD_reverse	unsigned char writedata2LCD_reverse(unsigned char number, unsigned char *value, unsigned char offset)	void writedata2LCD_reverse(unsigned char number, unsigned char *value, unsigned char offset)
writedata2LCD_normal	unsigned char writedata2LCD_normal(unsigned char number, unsigned char *value, unsigned char offset)	void writedata2LCD_normal(unsigned char number, unsigned char *value, unsigned char offset)
	inputs: number [1..255], a matrix beginning at pointer *value, offset **outputs:** character position at which an error occured (1..number of characters) or zero if no error occured **description:** These two procedures print a string on the LCD. Inputs are the pointer to the string and the length of the string. For example, the string «INFINEON» is an 8 character string. **comments:** writedata2LCD_normal() prints from left to right, while writedata2LCD_reverse() prints backwards(from right to left). **see also:** *write_char2LCD()*	

	non-optimized:	*optimized:*
send_instruction2LCD	unsigned char send_instruction2LCD(void)	void send_instruction2LCD(void)
	inputs: -	
	outputs: 0 if success 1 if there was an error (could not send an instruction to the LCD)	-
	description: This command sends the "Co RS RW 0 0 0 0 0" control byte to the slave LCD.	
	comments: This procedure is not used in this application note's code. The instructions first_ and second_instruction_group() are prefered instead. In addition, bit Co is always written as zero because this proved to be more efficient. However there is some possibility that another application could benefit from the use of Co as 1. The code size would then be reduced.	
	see also: *first_instruction_group(), second_instruction_group()*	

2.3 Discussion of the code-size and the optimization

There are two possible ways to optimize the code:

- disable any code related to I²C error checking/returning. For example, instead of using *unsigned char procedure(..)*, one could use *void procedure(..)*:

```
unsigned char write_char2LCD(unsigned char character,unsigned char offset)
{
if(I2cMasterWrite(character+offset)) return(1);
return(0);
}
```

This can be replaced by

```
void write_char2LCD(unsigned char character,unsigned char offset)
{
I2cMasterWrite(character+offset);
}
```

Also, instead of executing a

```
while(I2cMasterWrite(byte));     // send byte until an acknowledge is received
```

one could simply do a

```
I2cMasterWrite(byte);
```

discarding the output from this procedure.

- replace a procedure call with the procedure's code. As an example, instead of having

  ```
  write_char2LCD(0x7A,0x80);
  ```

  it is better to have this:

  ```
  I2cMasterWrite(0x7A+0x80);
  ```

When using an LCD, the code size of the procedures related to LCD controlling is usually critical. Using the non-optimized procedures may indeed provide communication error-detection, but the difference in code size should be set into consideration. From the author's point of view, one should use the optimized routines because the memory required is much smaller.

An error during the I²C initialization results in a blank Display. An error during runtime (and after a succesfull initialization) is not important; execution continues and the display will output text as soon as the I²C error is corrected. It is also not very hard to assure error-free communication by checking the hardware connections (and perhaps run a very simple test-LCD program, see §1.6).

To conclude, any communication errors need not to be identified and possibly reported because a LCD is not a critical component and the code-size is significant in most applications.

Code-size is important. This is why all visual effects in the next chapter are code size-optimized. In the code, the non-optimized procedures are provided as remarks next to the optimized ones. Also, excluding (deleting) unused procedures is very important because this will reduce the code size.

How much is the code size? The first lines of the link results are presented below:

```
BL51 BANKED LINKER/LOCATER V3.70c               25/10/99  15:32:48  PAGE 1


MS-DOS BL51 BANKED LINKER/LOCATER V3.70c, INVOKED BY:
C:\PROGLOC\MCU\C51PK\BIN\BL51.EXE C:\USERDATA\SKRAPA~1\FINAL_~1\I2C_SW8B.OBJ,
>> C:\USERDATA\SKRAPA~1\FINAL_~1\I2C_LCD.OBJ TO C:\USERDATA\SKRAPA~1\FINAL_~1\
>> I2C IX RS (256) PL (68) PW (78) CO (1000H) REGFILE (I2C.REG)


MEMORY MODEL: SMALL


INPUT MODULES INCLUDED:
  C:\USERDATA\SKRAPA~1\FINAL_~1\I2C_SW8B.OBJ (I2C_SW8B)
  C:\USERDATA\SKRAPA~1\FINAL_~1\I2C_LCD.OBJ (I2C_LCD)
  C:\PROGLOC\MCU\C51PK\LIB\C51S.LIB (?C_STARTUP)
  C:\PROGLOC\MCU\C51PK\LIB\C51S.LIB (?C?CLDOPTR)
  C:\PROGLOC\MCU\C51PK\LIB\C51S.LIB (?C?IMUL)


LINK MAP OF MODULE:  C:\USERDATA\SKRAPA~1\FINAL_~1\I2C (I2C_SW8B)
```

```
            TYPE    BASE      LENGTH    RELOCATION   SEGMENT NAME
            -----------------------------------------------------

            * * * * * * *   D A T A   M E M O R Y   * * * * * * *
            REG     0000H     0008H     ABSOLUTE     "REG BANK 0"
            DATA    0008H     0005H     UNIT         ?DT?_WRITEDATA2LCD_REVERS
>> E?I2C_LCD
            DATA    000DH     0002H     UNIT         ?DT?I2C_SW8B
                    000FH     0011H                  *** GAP ***
            BIT     0020H.0   0002H.2   UNIT         _BIT_GROUP_
                    0022H.2   0000H.6                *** GAP ***
            DATA    0023H     001AH     UNIT         _DATA_GROUP_
            IDATA   003DH     0001H     UNIT         ?STACK

            * * * * * * *   C O D E   M E M O R Y   * * * * * * *
            CODE    0000H     0003H     ABSOLUTE
                    0003H     0FFDH                  *** GAP ***
            CODE    1000H     0531H     INBLOCK      ?PR?MAIN?I2C_LCD
            CODE    1531H     0119H     INBLOCK      ?PR?TEXTEFFECT?I2C_LCD
            CODE    164AH     0104H     UNIT         ?CO?I2C_LCD
            CODE    174EH     00A6H     INBLOCK      ?PR?CREATE_FONT?I2C_LCD
            CODE    17F4H     000CH     UNIT         ?C_C51STARTUP
            CODE    1800H     00C9H     INBLOCK      ?PR?PRINTINFINEONLOGO?I2C
>> _LCD
            CODE    18C9H     00C3H     INBLOCK      ?PR?_TEXTROLL?I2C_LCD
            CODE    198CH     0070H     INBLOCK      ?PR?I2CSTOP?I2C_SW8B
            CODE    19FCH     006DH     INBLOCK      ?PR?_TEXTTRANS?I2C_LCD
            CODE    1A69H     006AH     INBLOCK      ?PR?_I2CMASTERWRITE?I2C_S
>> W8B
            CODE    1AD3H     005BH     INBLOCK      ?PR?_I2CMASTERREAD?I2C_SW
>> 8B
            CODE    1B2EH     0041H     UNIT         ?C?LIB_CODE
            CODE    1B6FH     003AH     INBLOCK      ?PR?INITIALIZE_LCD?I2C_LC
>> D
            CODE    1BA9H     002DH     INBLOCK      ?PR?_WRITEDATA2LCD_REVERS
>> E?I2C_LCD
            CODE    1BD6H     002BH     INBLOCK      ?PR?_WRITEDATA2LCD_NORMAL
>> ?I2C_LCD
            CODE    1C01H     0029H     INBLOCK      ?PR?I2CSTART?I2C_SW8B
            CODE    1C2AH     0021H     INBLOCK      ?PR?SEND_INSTRUCTION2LCD?
>> I2C_LCD
            CODE    1C4BH     001EH     INBLOCK      ?PR?CHECK_SCL?I2C_SW8B
```

```
            CODE    1C69H     001DH     INBLOCK       ?PR?DISPLAY_CONTROL_LCD?I
>> 2C_LCD
            CODE    1C86H     001BH     INBLOCK       ?PR?I2CINIT?I2C_SW8B
            CODE    1CA1H     0019H     INBLOCK       ?PR?CURSOR_DISPLAY_SHIFT_
>> LCD?I2C_LCD
            CODE    1CBAH     0018H     INBLOCK       ?PR?_WAIT2?I2C_LCD
            CODE    1CD2H     0013H     INBLOCK       ?PR?ENTRY_MODE_SET_LCD?I2
>> C_LCD
            CODE    1CE5H     0012H     INBLOCK       ?PR?FIRST_INSTRUCTION_GRO
>> UP?I2C_LCD
            CODE    1CF7H     0012H     INBLOCK       ?PR?SECOND_INSTRUCTION_GR
>> OUP?I2C_LCD
            CODE    1D09H     000DH     INBLOCK       ?PR?CLS_LCD?I2C_LCD
            CODE    1D16H     000DH     INBLOCK       ?PR?_GOTOXY_LCD?I2C_LCD
            CODE    1D23H     000BH     INBLOCK       ?PR?_WAIT?I2C_LCD
            CODE    1D2EH     000AH     INBLOCK       ?PR?_SET_CGRAM_ADDRESS_LC
>> D?I2C_LCD
            CODE    1D38H     0008H     INBLOCK       ?PR?CHECKCLOCK?I2C_SW8B
            CODE    1D40H     0008H     INBLOCK       ?PR?_SET_DDRAM_ADDRESS_LC
>> D?I2C_LCD
            CODE    1D48H     0007H     INBLOCK       ?PR?_WRITE_CHAR2LCD?I2C_L
>> CD
            CODE    1D4FH     0006H     INBLOCK       ?PR?RETURNHOME_LCD?I2C_LC
>> D
            CODE    1D55H     0006H     INBLOCK       ?PR?FUNCTION_SET_LCD?I2C_
>> LCD
```

I²C bus interface routines are marked with grey, while the very basic procedures for the I²C LCD control are marked with yellow. This gives us a very rough approximation of the code size:

I²C bus interface takes about 212 bytes, while the procedures used to control and print text on the LCD have a code length of about 238 bytes. Please note that these sizes are not the absolute minimum: they can be decreased. The number of procedures that is needed depends on the application. Their size depends on the level of manual optimization performed by the programmer.

3 The implementation of visual effects

Visual effects can now be developed by simply using the basic and standard procedures from the previous chapter. The reader is encouraged to read through the code, so as to fully understand the effects and modify them according to his/her needs.

In some of the procedures presented in this paper, offset is used. A suggested value for offset is 0x80 – when using compiler-defined ASCII strings. This is because ASCII and the LCD character set are different; normal ASCII characters correspond to LCD character set C characters by adding 0x80 (see the character set table in the PCF2116 datasheet). However, this is used only for testing. A final application could read the strings from an eeprom; or the strings could be stored with their correct values according to the LCD character set. offset should then be deleted from the procedures, saving code size.

The sample effects are:

3.1 texteffect

void texteffect(bit height, unsigned char *text1, unsigned char *text2, unsigned char length, unsigned char pos, char direction, unsigned char clearchar, unsigned char offset)

description:

This procedure prints out with a fade-in effect (with a direction of right, if direction=1 or left, if direction=0) a string or two strings at position pos on the display.

height is the number of strings minus one (:0 for one string and 1 for two strings) and text1, text2 are the matrices that contain the strings. Length is the length of each string (they must have the same length).

clearchar is the character from which the fade-in begins. A standard value is the space character(0xA0) and other interesting values are 0xA4 or 0x3D.

offset has been described before. It is added to each character of the two texts.

One could experiment by modifying the code in order to achieve faster or slower printing (change the values of maxmuzzy2 and the factor 10 at muzzystep2 - see the start of the procedure)

3.2 texttrans

void texttrans(unsigned char *text1, unsigned char length1, unsigned char *text2, unsigned char length2, unsigned char pos, unsigned char maxmuzzy3, unsigned char muzzystep3, interval, unsigned char offset)

description:

This effect is a transmission between two texts. The first text fades out while the second fades in at the same place. Transitional speed is defined by the values of maxmuzzy3 and muzzystep3.

3.3 textroll

void textroll(unsigned char *text, unsigned char length, unsigned char pos, interval , unsigned char rolltop, unsigned char offset)

description:

This effect is like the train-station boards that announce the departure and time-schedule of trains

Each character starts from the rolltop character and is decreased until it reaches the corresponding character from *text. Any space characters (defined as character_space) are ignored.

interval controls the speed of this effect.

It would be interesting to develop a similar but more complicated procedure like

void textroll2(unsigned char *text1, unsigned char *text2, unsigned char length, unsigned char pos, interval, unsigned char offset)

description: a line continually prints characters. Each character starts from the corresponding character from text1 and is decreased until it reaches the desired character (from text2), with a possible wrap-around (characters 0xA0 and 0xFF).

Suggested offset is 0x80.

3.4 printinfineonlogo

// special characters are defined and arranged in the lcd.h file

void create_font(void)

void printinfineonlogo(bit direction, bit clearchar, bit displaycursor, bit clr)

description:

This effect combines the use of special characters, texteffect and the cursor. It prints out the Infineon logo. This is a nice effect to implement before entering a sleep or idle mode.

direction and clearchar are defined in texteffect.

displaycursor is either true or false (if one desires the cursor to be printed during predefined intervals or not)

bit clr instructs to clear the screen before printing the logo (if true).

4 Epilogue

As described before, this application note can be used as a start when designing new LCD applications. Examples of such applications are portable instruments, home appliances and public information displays.

Appendix

The full executable code together with an I2C-bus module is found in the zip file that accompanies this application note. The C-compiler used was Keil 8051 v5.20 but one could of course use any C-compiler. This code was originally written for the Infineon C504 microcontroller. If one decides to use another microcontroller (example: Infineon C508), one should adapt the wait intervals inside and outside the effects in order to achieve normal speeds.

main() includes a demonstration of all the visual effects.

A

```
/*****************************************************************************/
/* The include file name: I2C_8b.DEF                                         */
/*                                                                           */
/* This is an include file for I/O port declaration of SCL and SDA           */
/* The users can modify this include file according to the I/O port           */
/* assignment of their preference.                                           */
/*                                                                           */
/*****************************************************************************/

sbit SCL = 0x90^0;    /* Port 1.0*/
sbit SDA = 0x90^1;    /* Port 1.1*/

//sbit SCL = 0xf8^0;    /* Port 1.0*/
//sbit SDA = 0xf8^1;    /* Port 1.1*/
```

B

```
/*****************************************************************************/
// lcd.h
//
// LCD configuration, constant definitions for the visual effects, LCD graphics
//
// An include file for the I2C_LCD.C program
/*****************************************************************************/


#define LCD_device_address 0x74       // LCD slave address: 01110100
#define USE_4_LINES              // Is it a 1, 2 or 4-line LCD?
#define G 1                 // See remark below

#define DL 1           // i2c interface to the lcd is used, so DL is predefined as 1
```

```
// These values define the speed of some of the effects
#define maxmuzzy 1000            //1300
#define muzzystep 10             //20

// Used with the wait() procedure
#define for_10_s 65535           //maximum wait
#define for_2_ms 580

#define nocursor 0
#define cursor 1
#define clearscreen 1
#define noclearscreen 0

// if G=1, use only one external voltage supply
// if G=0, use two external voltage supplies
// Please refer to the following schematics. They are used to control LCD contrast..
/*

Circuit diagram for the single LCD supply (bit G=1):

            Vdd +3.6..+6V
             ^
       100nF |
Vlcd----||---+----+
             |    |
             O    |
  V0------+->O    -- 100nF
             O    --
     100KOhm |    |
        pot  |    |
             |    |
             _    _
            Vss
```

```
Circuit diagram for the double LCD supply (bit G=0):

            Vdd +2.5..+6V
             ^
             |
  V0---------+
             |
       |     O    10K
 about |     O  resistor
 6.5V  |     O
       v     |
             |
             O    5K
Vlcd------+->O  potentiometer
             O
             |
             |
             |
             |
             ^
          -1V..-5V

*/


// The infineon logo
// These can be read from an eeprom!


// the following are used to create the font

unsigned char code logo0[8]={0x00,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C};
unsigned char code logo1[8]={0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C};
unsigned char code logo2[8]={0x00,0x00,0x00,0x00,0x00,0x00,0x1E,0x1B};
unsigned char code logo3[8]={0x1B,0x1B,0x1B,0x1B,0x1B,0x1B,0x1B,0x1B};
unsigned char code logo4[8]={0x00,0x07,0x0F,0x0C,0x0C,0x0C,0x1F,0x0C};
unsigned char code logo5[8]={0x00,0x00,0x00,0x0C,0x0C,0x00,0x0C,0x0C};
unsigned char code logo6[8]={0x00,0x00,0x00,0x00,0x00,0x00,0x0E,0x1B};
unsigned char code logo7[8]={0x1B,0x1B,0x1F,0x18,0x1B,0x1B,0x0E,0x0E};
unsigned char code logo8[8]={0x00,0x00,0x00,0x00,0x00,0x00,0x0E,0x1B};
unsigned char code logo9[8]={0x1B,0x1B,0x1B,0x1B,0x1B,0x1B,0x0E,0x0E};
unsigned char code logo10[8]={0x00,0x00,0x00,0x0C,0x1E,0x1E,0x0C,0x00}; // the dot above I
```

```
/* Some defined characters which are not used
unsigned char logo11[8]={0x1F,0x1F,0x0F,0x0F,0x1F,0x1F,0x0C,0x00};
unsigned char logo12[8]={0x1F,0x1F,0x1F,0x0F,0x1F,0x1F,0x0C,0x00};
unsigned char logo13[8]={0x1F,0x1F,0x1E,0x0F,0x1F,0x1F,0x0C,0x00};
unsigned char logo14[8]={0x1F,0x1F,0x1D,0x0F,0x1F,0x1F,0x0C,0x00};
unsigned char logo15[8]={0x1F,0x1F,0x1C,0x0F,0x1F,0x1F,0x0C,0x00};
unsigned char logo16[8]={0x1F,0x1F,0x1B,0x0F,0x1F,0x1F,0x0C,0x00};
*/

// the following are used to construct the logo (arrange the previous defined fonts)
unsigned char code logo_0_0[8]= {0x00,0x02,0x04,0x05,0x02,0x06,0x08,0x02};
unsigned char code logo_0_1[8]= {0x01,0x03,0x01,0x01,0x03,0x07,0x09,0x03};



/*
unsigned char logo_0=0x00; //00000
unsigned char logo_1= 0x0C; //01100
unsigned char logo_2= 0x1E; //11110
unsigned char logo_3= 0x1B; //11011
unsigned char logo_4= 0x07; //00111
unsigned char logo_5= 0x0F; //01111
unsigned char logo_6= 0x1F; //11111
unsigned char logo_7= 0x0E; //01110
unsigned char logo_8= 0x18; //11000

#define logo_0 0x00 //00000
#define logo_1 0x0C //01100
#define logo_2 0x1E //11110
#define logo_3 0x1B //11011
#define logo_4 0x07 //00111
#define logo_5 0x0F //01111
#define logo_6 0x1F //11111
#define logo_7 0x0E //01110
#define logo_8 0x18 //11000

I        n        f        i        e        o
00000         00000           00111       00000
01100         00000           01111       00000
01100         00000           01100       01100
01100         00000           01100       01100
01100         00000           01100       00000
01100         00000           11111       00000
01100           11110           01100           01100           01110           01110
```

```
01100         11011          01100          01100          11011          11011
01100         11011          01100          01100          11011          11011
01100         11011          01100          01100     11111         11011
01100         11011          01100          01100          11000          11011
01100         11011          01100          01100          11011          11011
01100         11011          01100          01100          11011          11011
01100         11011          01100          01100     01110          01110
*/
```

C

```
/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/*                  INFINEON TECHNOLOGIES Standard Software                  */
/*                                                                           */
/*___________________________________________________________________________*/
/*                                                                           */
/*       Programmer:   Dimitrios Skraparlis                                  */
/*       Department:   iE Munich                                             */
/*       Revision  :   2.0                                                   */
/*___________________________________________________________________________*/
/*                                                                           */
/*    Description: This program has the appropriate procedures to use        */
/*          LCD displays based on Philips PCF2116 family controllers         */
/*          using the i2c protocol.                                          */
/*                                                                           */
/*___________________________________________________________________________*/
/*                                                                           */
/*      Brief History:                                                       */
/*            10.08.1999: Start of the module                                */
/*            23.08.1999: Successful run                                     */
/*            25.08.1999: The first effects!                                 */
/*            08.09.1999: Beta version / cleaned up the code a little        */
/*            26.10.1999: Code is included in the application note.          */
/*___________________________________________________________________________*/
/*                                                                           */
/* Comments:                                                                 */
/*                                                                           */
```

```
/* The usage of [Co] can speed-up the code a little. It is not however used,     */
/* in order to produce general code. In other words, the code can be             */
/* optimised by carefully using Co and not always I2cstart..                     */
/*                                                                               */
/* Bear in mind that this code is not as optimised as it could be (in order      */
/* for it to be an excellent reference source).                                  */
/* It is also not optimized for readability.                                     */
/*********************************************************************************/

#include "lcd.h"
#include "i2c_sw8b.h"
#include "reg505L.h"



#define character_space 0x20 // the space character from character set C
//#define character_space 0xA0 // the second space character from character set C



#ifdef USE_4_LINES      // 4 lines * 12 characters
#define N 1
#define M 1
#define line1_start 0x00
#define line1_end 0x13
#define line2_start 0x20
#define line2_end 0x33
#define line3_start 0x40
#define line3_end 0x53
#define line4_start 0x60
#define line4_end 0x73
#define line_difference 0x20
#define line_length line_difference
#endif

#ifdef USE_2_LINES              // 2 lines * 24 characters
#define N 1
#define M 0
#define line1_start 0x00
#define line1_end 0x27
#define line2_start 0x40
#define line2_end 0x67
#define line_difference 0x40
```

```
#define line_length line_difference
#endif

#ifdef USE_1_LINE       // 1 line * 24 characters
#define N 0
#define M 0
#define line1_start 0x00
#define line1_end 0x4F
#define line_difference 0x00
#define line_length line_difference
#endif




#define DLNMG 2*G+4*M+8*N+0x10*DL+0x20



//*****************************************************************************************************
// Send an instruction to the LCD
//
// This command outputs the "Co RS RW 0 0 0 0 0" control byte to the slave LCD
//
//*****************************************************************************************************

void send_instruction2LCD(bit Co,bit RS,bit RW) //unsigned char send_instruction2LCD(bit Co,bit
RS,bit RW)


{
I2cMasterWrite(0x20*(unsigned char)(RW)|0x40*(unsigned char)(RS)|0x80*(unsigned char)(Co));
//   if (I2cMasterWrite(0x20*(unsigned char)(RW)|0x40*(unsigned char)(RS)|0x80*(unsigned char)(Co)))
return(1);
//   return(0);
}

//*****************************************************************************************************
//*
//* Enable access to the first LCD uC instruction group (transmit RS=0, RW'=0)
//*
//*****************************************************************************************************

void first_instruction_group(void)
{
```

```
I2cStart();
I2cMasterWrite(LCD_device_address);
send_instruction2LCD(0,0,0);
}

//*************************************************************************************************
//*
//* Enable access to the second LCD uc instruction group (transmit RS=1, RW'=0)
//*
//*************************************************************************************************
void second_instruction_group(void)
{
I2cStart();
I2cMasterWrite(LCD_device_address);
send_instruction2LCD(0,1,0);
}

//*************************************************************************************************
// Wait for a period of about (10 * time_out) cycles
//
// For the C504 with a 20MHz clock, 1 cycle=12/20480000 sec ~= 585.938 ns
// Therefore wait slows the uC down for (5.85938*time_out) usec
//
//*************************************************************************************************

void wait(time_out)
{
while (time_out--);
}

//*************************************************************************************************
// Example of a "wait for a much longer period" procedure
//
// With this procedure wait states of several seconds can be achieved
//*************************************************************************************************

void wait2(time_out)
{
while (time_out--) wait(time_out);
//unsigned int time_out2=time_out;
//while (time_out2--) {while (time_out--);}
}
```

```
//*************************************************************************************************
// Clear LCD Screen and wait until the LCD is ready (after about 2ms)
//
//   Screen is cleared, LCD shift is cancelled and cursor returns to position 0
//
//*************************************************************************************************

void cls_LCD(void) //unsigned char cls_LCD(void)

{
I2cMasterWrite(0x01);
wait(for_2_ms);
}


//*************************************************************************************************
// Return Home
//
//   Shift is cancelled and the cursor returns to position 0
//
// RS  R/W'
// --- ---
// 0   0
//*************************************************************************************************


void returnHome_LCD(void) //unsigned char returnHome_LCD(void)

{
I2cMasterWrite(0x02);
//   if (I2cMasterWrite(0x02)) return(1);       // send command: "return home"
//   return(0);
}


//*************************************************************************************************
// Set DDRAM address
//
// RS  R/W'
// --- ---
```

```
// 0   0
//*****************************************************************************************************


void set_DDRAM_address_LCD(unsigned char address) //unsigned char set_DDRAM_address_LCD(unsigned
char address)


{
I2cMasterWrite(address|0x80);
//   if (I2cMasterWrite(address|0x80)) return(1);
//   return(0);
}



//*****************************************************************************************************
// Set CGRAM address
//
// RS  R/W'
// --- ---
// 0   0
//*****************************************************************************************************


void set_CGRAM_address_LCD(unsigned char address) //unsigned char set_CGRAM_address_LCD(unsigned
char address)


{
I2cMasterWrite((address&0x7F)|0x40);
//   if (I2cMasterWrite((address&0x7F)|0x40)) return(1);
//   return(0);
}



//*****************************************************************************************************
// Write data backwards
// Inputs: *value : the start of the string
//         number : the string length
//
// Returns 0 if write was successful
// or the value of number +1 (i.e. 1 or 2 or 3.. etc) if not successful
// (this value indicates how many bytes were written +1)
// +1 is used because no written bytes would result in a return 0, which indicates successful write!
//
// Attention:
```

```
// One must specify Co=0 before this because here we send only data bytes. To send another control
byte,
// a I2cStart should be performed after this.
//
//
//*************************************************************************************************



void writedata2LCD_reverse(unsigned char number, unsigned char *value, unsigned char offset)
//unsigned char writedata2LCD_reverse(unsigned char number, unsigned char *value, unsigned char
offset)
{
second_instruction_group();


     while (number>0)
     {
     I2cMasterWrite(offset+*(value+number-1));
     //if(I2cMasterWrite(offset+*(value+number-1))) return(number+1); // if there is no
acknowledgement, abort and return the value of number
     number--;
        }
//return(0);
}



//*************************************************************************************************
// Write data normally (not backwards)
//
// Inputs: *value : the start of the string
//         number : the string length
//
// Returns 0 if write was successful
// or the value of counter +1 if not successful
// (this value indicates how many bytes were written +1)
// +1 is used because no written bytes would result in a return 0, which indicates successful write!
//
// Attention:
// One must specify Co=0 before this because here we send only data bytes. To send another control
byte,
// a I2cStart should be performed after this.
//
//
//*************************************************************************************************
```

```
void writedata2LCD_normal(unsigned char number, unsigned char *value, unsigned char offset)
//unsigned char writedata2LCD_normal(unsigned char number, unsigned char *value, unsigned char
offset)
{


/* Instead of this code:
     unsigned char lastnumber=number;
second_instruction_group();
     while (number>0)
     {
     if(I2cMasterWrite(offset+*(value+lastnumber-number))) return(lastnumber-number); // if there is
no acknowledgement, abort and return the value of number
     number--;
        }
     return(0);
   better use the following code:
*/
     unsigned char counter=0;
second_instruction_group();
     while (counter<number)
     {
     I2cMasterWrite(offset+*(value+counter));
     //if(I2cMasterWrite(offset+*(value+counter))) return(counter+1); // if there is no
acknowledgement, abort and return the value of number
     counter++;
        }
     //return(0);


}


//*****************************************************************************************************
// Write a character to the LCD
//
// RS  R/W'
// --- ---
// 1   0
//
//*****************************************************************************************************


void write_char2LCD(unsigned char character,unsigned char offset) //unsigned char
write_char2LCD(unsigned char character,unsigned char offset)
{
I2cMasterWrite(character+offset);
//   if(I2cMasterWrite(character+offset)) return(1);
//   return(0);
```

```
}

//*************************************************************************************************
// Goto (X,line) on the LCD memory
//
//*************************************************************************************************

void gotoXY_LCD(unsigned char pos) //unsigned char gotoXY_LCD(unsigned char pos)

{
first_instruction_group();
I2cMasterWrite(0x80|pos);
//   if(I2cMasterWrite(0x80|pos)) return(1);
//   return(0);
}

//*************************************************************************************************
// Entry mode set
//
// RS  R/W'
// --- ---
// 0   0
//*************************************************************************************************

void entry_mode_set_LCD(bit ID,bit S) //unsigned char entry_mode_set_LCD(bit ID,bit S)

{
I2cMasterWrite(((unsigned char)(S))|(2*(unsigned char)(ID))|0x04);
//       if (I2cMasterWrite(((unsigned char)(S))|(2*(unsigned char)(ID))|0x04)) return(1);
//   return(0);
}

//*************************************************************************************************
// Display control
//
// RS  R/W'
// --- ---
// 0   0
//*************************************************************************************************

void display_control_LCD(bit D,bit C,bit Blink)  //unsigned char display_control_LCD(bit D,bit C,bit
Blink)
```

```
{
I2cMasterWrite(((unsigned char)(Blink))|(2*(unsigned char)(C))|(4*(unsigned char)(D))|0x08);
//   if (I2cMasterWrite(((unsigned char)(Blink))|(2*(unsigned char)(C))|(4*(unsigned
char)(D))|0x08)) return(1);
//   return(0);
}

//*****************************************************************************************************
// Cursor/display shift
//
// RS  R/W'
// --- ---
// 0   0
//*****************************************************************************************************

void cursor_display_shift_LCD(bit SC,bit RL) //unsigned char cursor_display_shift_LCD(bit SC,bit RL)

{
I2cMasterWrite((4*(unsigned char)(RL))|(8*(unsigned char)(SC))|0x10);
//   if (I2cMasterWrite((4*(unsigned char)(RL))|(8*(unsigned char)(SC))|0x10)) return(1);
//   return(0);
}

//*****************************************************************************************************
// Function set
//
// Note:
// the bits used inside(bit DL,bit N,bit M,bit G) are global constants//
//
// RS  R/W'
// --- ---
// 0   0
//*****************************************************************************************************

void function_set_LCD(void) //unsigned char function_set_LCD(void)

{
I2cMasterWrite(DLNMG);
//   if (I2cMasterWrite(DLNMG)) return(1);
//   return(0);
}
```

```
//*************************************************************************************************
//
// Initialize liquid crystal display
//
// Addresses the LCD slave, initializes the display and stops communication (with I2cStop)
//
// Note:
// DL, N, M ,G are constant - this is why they are not p as parametres
//*************************************************************************************************


void initialize_LCD(bit ID,bit S,bit D,bit C,bit Blink,bit SC,bit RL) //unsigned char
initialize_LCD(bit ID,bit S,bit D,bit C,bit Blink,bit SC,bit RL)


{
//wait(for_2_ms);                         // 2 ms wait for the LCD to start up
I2cStart();
I2cMasterWrite(LCD_device_address);
send_instruction2LCD(0,0,0);
function_set_LCD();
display_control_LCD(D,C,Blink);
entry_mode_set_LCD(ID,S);
cursor_display_shift_LCD(SC,RL);

/*        if (I2cMasterWrite(LCD_device_address))            // Address the slave
     {
        I2cStop();                          // No acknowledge? Then
        return(1);                          // stop and return 1.
        }

        if (send_instruction2LCD(0,0,0)) //Co,RS,RW
     {
        I2cStop();
        return(1);
        }

       if(function_set_LCD())
         {
        I2cStop();
        return(1);
     }

       if(display_control_LCD(D,C,Blink))
         {
```

```
        I2cStop();
        return(1);
    }

      if(entry_mode_set_LCD(ID,S))
        {
        I2cStop();
        return(1);
    }

      if(cursor_display_shift_LCD(SC,RL))
        {
        I2cStop();
        return(1);
    }

// I2cStop(); // It is not needed

return(0);
*/

}



//*************************************************************************************************
//* Write (bit: height)-line text smoothly at position pos.
//* If height=1 then output a two line text
//* If height=0 then output a one line text
//* Direction:
//* 1 for right, 0 for left
//*
//*************************************************************************************************
void texteffect(bit height, unsigned char *text1, unsigned char *text2, unsigned char length,
unsigned char pos, char direction, unsigned char clearchar, unsigned char offset)
{
unsigned int muzzy;
char counter;
unsigned char maxmuzzy2=1000;
unsigned char muzzystep2=10*((unsigned char)(height)+1);
unsigned char startcounter;
char counterstep;
if (direction==1) { startcounter=0; counterstep=1; length=length-1; direction++;}
else if (direction==0) { startcounter=length-1; counterstep=-1; length=0;}
```

```
for (counter=startcounter;(direction-1)*counter<=length;counter=counter+counterstep)
{
for(muzzy=0;muzzy<maxmuzzy2;muzzy=muzzy+muzzystep2)
{
     first_instruction_group();
     I2cMasterWrite(0x80|(pos+counter));//gotoXY_LCD(pos+counter);
     second_instruction_group();
     I2cMasterWrite(clearchar);//write_char2LCD(clearchar,0);

     if (height==1)
     {
     first_instruction_group();
     I2cMasterWrite(0x80|(pos+counter+line_length));//gotoXY_LCD(pos+counter+line_length);
     second_instruction_group();
     I2cMasterWrite(clearchar);//write_char2LCD(clearchar,0);
     }

wait(maxmuzzy2-muzzy);// This is a linear transition. Another possible value: maxymuzzy2/muzzy

     first_instruction_group();
     I2cMasterWrite(0x80|(pos+counter));//gotoXY_LCD(pos+counter);
     second_instruction_group();
     I2cMasterWrite(*(text1+counter)+offset);//write_char2LCD(*(text1+counter), offset);

     if (height==1)
     {
     first_instruction_group();
     I2cMasterWrite(0x80|(pos+counter+line_length));//gotoXY_LCD(pos+counter+line_length);
     second_instruction_group();
     I2cMasterWrite(*(text2+counter)+offset);//write_char2LCD(*(text2+counter), offset);
     }

wait(muzzy);

}
//wait(10000);
}
}


//*****************************************************************************************
//* Text transition from text1 to text2 at position pos.
```

```
//*
//*************************************************************************************************
void texttrans(unsigned char *text1, unsigned char length1, unsigned char *text2, unsigned char
length2, unsigned char pos, unsigned char maxmuzzy3, unsigned char muzzystep3,  interval, unsigned
char offset)
{


unsigned char muzzy;
for(muzzy=0;muzzy<maxmuzzy3;muzzy=muzzy+muzzystep3)
{
first_instruction_group();
I2cMasterWrite(0x80|pos);//gotoXY_LCD(pos);
second_instruction_group();
writedata2LCD_normal(length1,text1, offset);
wait(maxmuzzy3/muzzy); // This looks like an exponential transition. Another possible value:
maxymuzzy3-muzzy


first_instruction_group();
I2cMasterWrite(0x80|pos);//gotoXY_LCD(pos);
second_instruction_group();
writedata2LCD_normal(length2,text2, offset);
wait(muzzy);
}
wait2(interval);
}




//*************************************************************************************************
//* Roll text at pos
//*
//*************************************************************************************************


void textroll(unsigned char *text, unsigned char length, unsigned char pos, interval, unsigned char
rolltop, unsigned char offset)
{
unsigned char counter2;
unsigned char counter;
unsigned char min=rolltop;


for(counter=0;counter<length;counter++)
{
if ((*(text+counter)<min)&(*(text+counter)!=character_space)) min=*(text+counter); //calculate
minimum character code
                  //Do not mess with the space char: " "
```

```
}


for(counter2=rolltop;counter2>=(min+offset);counter2--)
{
          first_instruction_group();
          I2cMasterWrite(0x80|pos);//gotoXY_LCD(pos);
          second_instruction_group();

     for(counter=0;counter<length;counter++)
     {
          // Don't do this effect if space character: " "
          if ((*(text+counter)<counter2-offset)&(*(text+counter)!=character_space))
I2cMasterWrite(counter2);//write_char2LCD(counter2,0);
          else I2cMasterWrite(*(text+counter)+offset);//write_char2LCD(*(text+counter),offset);
     }
wait(interval);

}

}



//*****************************************************************************************************
//* Writes the user-defined characters to the CG-RAM
//*
//*****************************************************************************************************

void create_font(void)
{
first_instruction_group();
set_CGRAM_address_LCD(0x00);
second_instruction_group();
writedata2LCD_normal(8,logo0,0);
writedata2LCD_normal(8,logo1,0);
writedata2LCD_normal(8,logo2,0);
writedata2LCD_normal(8,logo3,0);
writedata2LCD_normal(8,logo4,0);
writedata2LCD_normal(8,logo5,0);
writedata2LCD_normal(8,logo6,0);
writedata2LCD_normal(8,logo7,0);
writedata2LCD_normal(8,logo8,0);
```

```
writedata2LCD_normal(8,logo9,0);
writedata2LCD_normal(8,logo10,0);
/*
writedata2LCD_normal(8,logo11,0);
writedata2LCD_normal(8,logo12,0);
writedata2LCD_normal(8,logo13,0);
writedata2LCD_normal(8,logo14,0);
writedata2LCD_normal(8,logo15,0);
writedata2LCD_normal(8,logo16,0);
*/
}

//*************************************************************************************************
//* Prints the infineon logo - good before idle mode entry
//*
//*************************************************************************************************

void printinfineonlogo(bit direction, bit clearchar, bit displaycursor, bit clr)
{
unsigned char dummy2[1];
I2cStart();
I2cMasterWrite(LCD_device_address);

first_instruction_group();

if(clr) // clear the screen?
{
cls_LCD();
wait(for_2_ms);
}

if(displaycursor)
{
display_control_LCD(1,0,0); //hide cursor
}

texteffect(1,logo_0_0,logo_0_1,8,2+line2_start,direction,clearchar,0);


if(displaycursor)
{
//show cursor
first_instruction_group();
```

```
gotoXY_LCD(10+line3_start);
display_control_LCD(1,1,1);
}

wait (65535);
wait (65535);

if(displaycursor)
{
//hide cursor
first_instruction_group();
display_control_LCD(1,0,0);
}

// SHOW DOT ABOVE I
*dummy2=0x0A;
texteffect(0,dummy2,0,1,2+line1_start,direction,0x20,0);

if(displaycursor)
{
//show cursor again
first_instruction_group();
gotoXY_LCD(10+line3_start);
display_control_LCD(1,1,1);
}

wait2 (500);
}
```

D

```
//*********************************
// DEMONSTRATION OF LCD ROUTINES
//
// Written by Dimitrios Skraparlis
//*********************************

#include "i2c_lcd.h"

unsigned char code string_spaces[6]="      ";
unsigned char code string_sample[6]="sample";
unsigned char code string_code[6]=" code ";
unsigned char code string_LCD[6]=" LCD  ";
```

```
//*************************************************************************************************
//*************************************************************************************************
//
//                          MAIN
//
//*************************************************************************************************
//*************************************************************************************************

void main (void)
{

unsigned int dummy0;
unsigned int dummy1;
unsigned int dummy2;
bit direction=1;

// Initialize display

// Configure bits used in the LCD driver instructions
// _____________________________________________________
// Entry mode set instruction
bit ID=1;          // decrement/increment
bit S=0;           // display freeze/shift
// Display control instruction
bit D=1;           // display off/on
bit C=1;           // cursor off/on
bit Blink=0;            // blink off/on
// Cursor/display shift instruction
bit SC=0;          // cursor move / display shift
bit RL=0;          // left/right shift
// Function set instruction
     /*The following are defined somewhere else in the start of this code; they are constants
       bit DL  // 4/8 bits (interface data length - i2c protocol is used here)
       bit N       // Selects type of display
       bit M       // Selects type of display
       bit G       // voltage generator Vlcd=Vo / Vlcd=Vo-0.8Vdd*/
//bit Co;          // indicates data bytes / data byte+another control byte  following
//bit RS;          // selects instruction
//bit RW;          // selects instruction
// _____________________________________________________
```

```
I2cInit();

/*
// ###############################################################################################
// ### Start of test - instructions send to lcd, according to the example in the datasheet.    ###

     I2cStart();                      //step 1
     I2cMasterWrite(0x74);//01110100);          //step 2 // SLAVE ADDRESS
     I2cMasterWrite(0x00);//00000000);     //step 3 // CONTROL BYTE
     I2cMasterWrite(0x3E);//00111110);     //step 4 // _FUNCTION SET        // * changed
     I2cMasterWrite(0x0E);//00001110);     //step 5 // _DISPLAY ON/OFF CONTROL
     I2cMasterWrite(0x06);//00000110);     //step 6 // _ENTRY MODE SET
     I2cStart();                      //step 7
     I2cMasterWrite(0x74);//01110100);     //step 8 // SLAVE ADDRESS
     I2cMasterWrite(0x40);//01000000);     //step 9 // CONTROL BYTE

     I2cMasterWrite(character_H);
     I2cMasterWrite(character_E);
     I2cMasterWrite(character_L);
     I2cMasterWrite(character_L);
     I2cMasterWrite(character_O);

     I2cStop();                       //(step 17)

// ### End of test                                          ###
// ###############################################################################################
*/


initialize_LCD(ID,S,D,C,Blink,SC,RL);



while(1) //loop endlessly
{

//**************
// CLEAR SCREEN
//**************
first_instruction_group();
cls_LCD();

//**********************
```

```
// CREATE INFINEON FONT
//**********************
create_font();

//**********************
// PRINT INFINEON LOGO
//**********************

printinfineonlogo(direction, dummy1, cursor, noclearscreen);
display_control_LCD(1,0,0); //hide cursor

wait2(150);


//************************
// PRINT INFINEON LOGO 2
//************************

first_instruction_group();
cls_LCD();

dummy1=0x5C;
dummy0=2; //loop 2 times
while(dummy0--)
{
first_instruction_group();
cls_LCD();

dummy1++;
direction=!direction;

texteffect(1,logo_0_0,logo_0_1,8,2+line1_start,direction,dummy1,0);
texteffect(0,"technologies","",12,0+line3_start,!direction,dummy1,0x80);
wait(65535);
wait(65535);
}

//**********************
// SHIFT DISPLAY
//**********************

first_instruction_group();
dummy2=20;
```

```
     while(dummy2--)
     {
     wait(29535);
     cursor_display_shift_LCD(1,1);
     }

wait (65535);
wait (65535);

//*********************
// CLEAR VISIBLE AREA
//*********************

texteffect(0,"            ","",12,0+line3_start,1,character_space,0);
texteffect(1,"            ","            ",12,0+line1_start,0,character_space,0);
wait (65535);


//************************************
// PRINT "infineon technologies Munich"
//************************************

texteffect(0,"infineon","",8,2+line1_start,1,character_space,0x80);
texteffect(0,"technologies","",12,0+line2_start,0,character_space,0x80);
texteffect(0,"MMI  IE SE","",10,1+line3_start,1,0x2E,0x80);


wait (65535);
wait (65535);
wait (65535);

gotoXY_LCD(line1_start+12);
writedata2LCD_normal(8,logo_0_0,0);

gotoXY_LCD(line2_start+12);
writedata2LCD_normal(8,logo_0_1,0);

gotoXY_LCD(line3_start+12);
writedata2LCD_normal(7,"Munich!",0x80);


//***************
// SHIFT DISPLAY
```

```
//***************

first_instruction_group();
dummy2=10;
     while(dummy2--)
     {
     wait(29535);
     cursor_display_shift_LCD(1,0);
     }

//***********************
// ERASE SOME CHARACTERS
//***********************
gotoXY_LCD(line1_start+10);
writedata2LCD_normal(2,"  ",0);

gotoXY_LCD(line2_start+10);
writedata2LCD_normal(2,"  ",0);

gotoXY_LCD(line3_start+10);
writedata2LCD_normal(2,"  ",0);

gotoXY_LCD(20+line1_start);
writedata2LCD_normal(2,"  ",0);

gotoXY_LCD(20+line2_start);
writedata2LCD_normal(2,"  ",0);

gotoXY_LCD(20+line3_start);
writedata2LCD_normal(2,"  ",0);

wait (65535);
wait (65535);
wait (65535);

//**************
// CLEAR SCREEN
//**************
first_instruction_group();
cls_LCD();

//*****************************
// PRINT "Industrial Design iE"
```

```
//*****************************

wait(65535);
wait(65535);
wait(65535);
wait(65535);
texteffect(1,"Ü NDUSTRIAL ","E LECTRONICS",12,0+line1_start,0,0x3D,0x80);
wait(65535);
texteffect(0,"E LECTRONICS","",12,0+line3_start,1,0x3D,0);
wait(65535);
wait(65535);
wait(65535);


//*******************************************
// PRINT "infineon technologies - sample code"
//*******************************************

first_instruction_group();
cls_LCD();

texttrans("        ",8,"infineon",8,2+line1_start,maxmuzzy,muzzystep,500,0x80);
texttrans("            ",12,"technologies",12,0+line2_start,maxmuzzy,muzzystep,500,0x80);

wait(65535);
wait(65535);

texttrans(string_spaces,6,string_sample,6,3+line3_start,maxmuzzy,muzzystep,500,0x80);
dummy1=2;
while(dummy1--)
{

texttrans(string_sample,6,string_code,6,3+line3_start,maxmuzzy,muzzystep,500,0x80);
texttrans(string_code,6,string_sample,6,3+line3_start,maxmuzzy,muzzystep,500,0x80);
}
texttrans(string_sample,6,string_LCD,6,3+line3_start,maxmuzzy,muzzystep,500,0x80);
texttrans(string_LCD,6,string_code,6,3+line3_start,maxmuzzy,muzzystep,500,0x80);

wait2(500);
```

```
//************************************************
// PRINT by rolling "Siemens uC applying i2C LCD control."
//************************************************

first_instruction_group();
cls_LCD();

wait(65535);

// SECOND EFFECT
textroll("Infineon  uC",12,0+line1_start,15000,0xFF,0x80);
//wait(65535);
textroll("applying i2C",12,0+line2_start,13000,0xFF,0x80);
//wait(65535);
textroll("LCD control.",12,0+line3_start,11000,0xFF,0x80);
wait(65535);
wait(65535);
wait(65535);


} // end of LOOP
} // end of main
```

E The following code is the I²C bus interface code, taken from the Infineon 8 bit microcontrollers' application note AP083701.

```
/*****************************************************************************/
/*                                                                           */
/*                      SIEMENS Standard Software                            */
/*                                                                           */
/*                      Unauthorized copying prohibited                      */
/*                                                                           */
/*===========================================================================*/
/*       Programmer:   Sylvia Gusowski                                       */
/*       Department:   Siemens HL CC AT                                      */
/*       Revision  :   1.0                                                   */
/*                                                                           */
/*===========================================================================*/
/*       Description:  This module is a standard I2c bus single master       */
/*                 prococol by using CPU time.                               */
/*                                                                           */
/*      The clock frequency is approximately 100kHz with 20 MHz CPU clock    */
/*   and 300 ns for a one-cycle instruction. The timing of the clock pulses  */
/*   are controlled by using NOPs.                                           */
```

```
/*                                                                          */
/*   Subroutines can be called from main program: I2cInit, I2cStart,        */
/*   I2cMasterWrite, I2cMasterRead, I2cStop.                                */
/*                                                                          */
/*   Requirement: In application program, the include files called          */
/*   I2C_SW8b.H and I2C_8b.DEF must be attached to the program. In the      */
/*   I2C.DEF file, the variable SDA and SCL must be declared as any         */
/*   two of the I/O pins.                                                   */
/*                                                                          */
/*   In case there is a fault on the bus, for instance, the SDA line is     */
/*      shorted to ground or pulled down to LOW by the slave device. This   */
/*      module will generate clock pulses until the line is released and the*/
/*      time-out is 10 ms before returing a "HIGH" value from the I2cInit or*/
/*      I2cStop. For SCL line, it will monitor until the line is released and*/
/*      the time-out is 10 ms. The return value of I2cInit or I2cStop will be*/
/*      checked in the main program to take an appropriate action.          */
/*                                                                          */
/*      Input and output parameters of those subroutines:                   */
/*      (1) I2cInit: no input                                               */
/*                output = "0" - no error, "1" - error                      */
/*                                                                          */
/*      (2) I2cStart: no input, no output                                   */
/*                                                                          */
/*      (3) I2cMasterWrite: input = one byte of data to be sent to slave    */
/*                       device.                                            */
/*                  output = acknowledge bit.                               */
/*                            (0 - received, 1 - not received)              */
/*                                                                          */
/*   (4) I2cMasterRead: input = "1"/"0" of acknowledge bit to slave device  */
/*                 (send on 9th clock pulse of data received)               */
/*                           output = A byte of data from slave device.     */
/*                                                                          */
/*   (5) I2cStop: no input                                                  */
/*              output = "0" - no error, "1" - error                        */
/*                                                                          */
/*==========================================================================*/
/*      History:                                                            */
/*            05/02/99: Start of the module                                 */
/*                                                                          */
/****************************************************************************/

#include <reg51.h>
#include <intrins.h>
```

```
#include "i2c_8b.def"

#define NOP _nop_();
#define Delay1 NOP
#define Delay16 { NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP NOP}
#define period 2900         /* approximately 10 ms time out for bus faulty */

unsigned int time_out;


void CheckClock();
unsigned char Check_SCL();
unsigned char I2cInit();
void I2cStart();
unsigned char I2cMasterWrite(unsigned char input_byte);
unsigned char I2cMasterRead(unsigned char ack);
unsigned char I2cStop();

/*****************************************************************************/
/*   Subroutine:   CheckClock                                                */
/*                                                                           */
/*   Description:  Send HIGH and read the SCL line. It will wait until       */
/*            the line has been released from slave device for               */
/*            every bit of data to be sent or received.                      */
/*                                                                           */
/*      Input:     None                                                      */
/*                                                                           */
/*      Return:         None                                                 */
/*                                                                           */
/*****************************************************************************/

void CheckClock()
{



        while (!SCL)           /* check for wait state before sending or */
                  /* receiving any data. */
        SCL = 1;



}
```

```
/*****************************************************************************/
/*   Subroutine:    Check_SCL                                                */
/*                                                                           */
/*   Description:  Send HIGH and read the SCL line. It will wait until       */
/*            the line has been released from slave device with the          */
/*            time out of approximately 10 ms (20 MHz CPU and 300 ns         */
/*                      for a one-cycle instruction).                        */
/*                                                                           */
/*      Input:     None                                                      */
/*                                                                           */
/*      Return:         "0" - SCL line is OK                                 */
/*            "1" - SCL line is faulty                                       */
/*                                                                           */
/*****************************************************************************/

unsigned char Check_SCL()
{

     time_out = period;



     while (time_out--)
     {
        if (!SCL)       /* wait if SCL is pulled down to LOW by slave device */
        {
              SCL = 1;          /* set clock to high */

           return (0);
        }
          }

     return (1);        /* ERROR: SCL line is stuck to low */

}
```

```
/*****************************************************************************/
/*   Subroutine:   I2cInit                                                   */
/*                                                                           */
/*   Description:  Initialize the I2C bus                                    */
/*                                                                           */
/*      Input:     None                                                      */
/*                                                                           */
/*      Return:         "0" - bus line is OK                                 */
/*            "1" - bus line is faulty                                       */
/*                                                                           */
/*****************************************************************************/

unsigned char I2cInit()
{



     if (!SDA)          /* if lines are low, set them to high */
       if (I2cStop())
          return (1);

     if (!SCL)
       if (I2cStop())
          return (1);

     return (0);

}


/*****************************************************************************/
/*   Subroutine:   I2cStart                                                  */
/*                                                                           */
/*   Description:  Generate a START condition on I2C bus                     */
/*                                                                           */
/*      Input:     None                                                      */
/*                                                                           */
/*      Return:         None                                                 */
/*                                                                           */
/*****************************************************************************/

void I2cStart()
{
```

```
    SDA = 1;        /* to make sure the SDA and SCL are both high */
    SCL = 1;
       Delay16;

    SDA = 0;        /* SDA line go LOW first */
    Delay16;
    SCL = 0;        /* then followed by SCL line with time delay */

}


/*****************************************************************************/
/*   Subroutine:   I2cMasterWrite                                            */
/*                                                                           */
/*   Description:  Output one byte of data to slave device. Check for        */
/*            WAIT condition before every bit is sent.                       */
/*                                                                           */
/*      Input:     one byte of data to be sent to slave device.              */
/*                                                                           */
/*      Return:         acknowledgement from slave:                          */
/*                0 = acknowledge is received                                */
/*                  1 = no acknowledge is received                           */
/*                                                                           */
/*****************************************************************************/

unsigned char I2cMasterWrite(unsigned char input_byte)
{
    unsigned char mask,i;


       mask = 0x80;


    for (i=0; i<8; i++)         /* send one byte of data */
    {
       if (mask & input_byte)   /* send bit according to data */
          SDA = 1;
       else SDA = 0;

       mask = mask >> 1;        /* shift right for the next bit */
```

```
        Delay1;

           CheckClock();                  /* check SCL line */

        Delay16;

        SCL = 0;                      /* clock is low */

        Delay16;

        }

        SDA = 1;                /* release SDA line*/
     Delay1;

        SCL = 1;        /* generate 9th clock pulse */
        Delay16;
          mask = SDA;        /* read acknowledge */


     SCL = 0;                /* clock is low */

     Delay16;      /* to avoid short pulse transition on SDA line */

     return (mask);          /* return acknowledge bit */
}



/*************************************************************************** */
/*   Subroutine:   I2cMasterRead                                            */
/*                                                                          */
/*   Description:  Read one byte of data from the slave device. Check       */
/*            for WAIT condition before every bit is received.              */
/*                                                                          */
/*      Input:          Acknowledge require:                                */
/*            0 - generate LOW output after a byte is received              */
/*            1 - generate HIGH output after a byte is received             */
/*                                                                          */
/*      Return:    received one byte of data from slave device             */
/*                                                                          */
/*                                                                          */
/*************************************************************************** */
```

```
unsigned char I2cMasterRead(unsigned char ack)
{

     unsigned char mask,i,rec_data;


     rec_data = 0;
     mask = 0x80;


     for (i=0; i<8; i++)
     {


            CheckClock();        /* clock is high */

            if (SDA)                    /* read data while clock is high */
          rec_data |= mask;

         mask = mask >> 1;

         SCL = 0;                    /* clock is low */
            Delay16;
     }


     if (ack)               /* set SDA data first before port direction */
        SDA = 1;             /* send acknowledge */
     else SDA = 0;

     Delay1;

     SCL = 1;           /* clock is high */
     Delay16;

     SCL = 0;                /* clock is low */

        SDA = 1;
        Delay16;        /* to avoid short pulse transition on SDA line */

     return (rec_data);
}
```

```
/*****************************************************************************/
/*   Subroutine:   I2cStop                                                   */
/*                                                                           */
/*   Description:  generate stop condition on the I2C bus                    */
/*                                                                           */
/*      Input:          none                                                 */
/*                                                                           */
/*      Return:    "0" - the bus line is OK                                  */
/*                 "1" - the bus line has been pulled down to low            */
/*                                                                           */
/*****************************************************************************/

unsigned char I2cStop()
{

     time_out = period;
     while (time_out --)
     {
        if (!SDA)                    /* check SDA line */
        {
              SCL = 1;               /* generate a clock pulse if SDA is pull */
          Delay16;      /* down to low */
          SCL = 0;
          Delay16;
        }
        else                     /* check SCL line */
           {
              SDA  = 0;
                Delay1;

          if (Check_SCL())       /* to generate STOP condition */
             return (1);         /* ERROR: SCL line is stuck to low */

              Delay16;

          SDA = 1;
          Delay16;
          return (0);
       }
        }

        return (1);                  /* ERROR: SDA line is stuck to low */
}
```