SIEMENS

Standard EEPROM ICs

Interfacing SLx 24Cxx I^2C-Bus
Serial EEPROMs to 8051 Controller Family,
especially to the Siemens C500 Controller Family

Application Note 1999-03-15

Interfacing SLx 24Cxx, Application Note **Revision History: Current Version: 1999-03-15**		
Previous Version:		
Page (in previous Version)	Page (in current Version)	Subjects (major changes since last revision)

Page Protection Mode™ is a trademark of Siemens AG.

Edition 1999-03-15

Published by Siemens AG,
Bereich Halbleiter, Marketing-
Kommunikation, Balanstraße 73,
81541 München

Attention please!

As far as patents or other rights of third parties are concerned, liability is only assumed for components, not for applications, processes and circuits implemented within components or assemblies.

The information describes the type of component and shall not be considered as assured characteristics.

Terms of delivery and rights to change design reserved.

For questions on technology, delivery and prices please contact the Semiconductor Group Offices in Germany or the Siemens Companies and Representatives worldwide (see address list).

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Siemens Office, Semiconductor Group.

Siemens AG is an approved CECC manufacturer.

Packing

Please use the recycling operators known to you. We can also help you – get in touch with your nearest sales office. By agreement we will take packing material back, if it is sorted. You must bear the costs of transport.

For packing material that is returned to us unsorted or which we are not obliged to accept, we shall have to invoice you for any costs incurred.

Components used in life-support devices or systems must be expressly authorized for such purpose!

Critical components[1] of the Semiconductor Group of Siemens AG, may only be used in life-support devices or systems[2] with the express written approval of the Semiconductor Group of Siemens AG.

1 A critical component is a component used in a life-support device or system whose failure can reasonably be expected to cause the failure of that life-support device or system, or to affect its safety or effectiveness of that device or system.

2 Life support devices or systems are intended (a) to be implanted in the human body, or (b) to support and/or maintain and sustain human life. If they fail, it is reasonable to assume that the health of the user may be endangered.

Table of Contents **Page**

1 Introduction

A lot of applications in the field of consumer electronics, telecommunications, automotive and industrial electronics need non-volatile memory. Because of their byte level flexibility, low I/O pin requirements, low power consumption, small footprint and low cost, Siemens serial EEPROMs are a popular choice for non-volatile storage.

This application note provides assistance and source code to ease the design process of interfacing a SLx 24Cxx I²C-bus serial EEPROM to the 8051 controller family, especially to the Siemens C500 8-bit controller family. The source code (`24Cxx8b.EXE`) as well as the datasheets for SLx 24Cxx EEPROMs are also available in the Siemens World Wide site at address:

http://www.siemens.de/semiconductor/products/ics/31/316.htm.

2 I²C-Bus

SLx 24Cxx serial EEPROMs use the industry standard I²C-bus protocol for data transfer. The I²C-bus or Inter-Integrated Circuit bus has been developed by Philips. It allows integrated circuits to communicate directly with each other via a simple bi-directional 2-wirebus. The two bus lines are serial clock line (SCL), and serial data line (SDA).

Nowadays, the I²C-bus becomes a standard bus system which is used in consumer electronics, telecommunications and industrial electronics. This software for interfacing a SLx 24Cxx I²C-bus serial EEPROM to the 8051 controller family by I²C-bus emulation supports the single master protocol only. It is using the CPU time to generate clock, and transmit or receive the data. The clock frequency of the I²C-bus can achieve up to 400 kHz.

2.1 Data Transfer Formats

The I²C-bus protocol allows several devices to share the same bus. In this software the microcontroller serves as the master, initiating and terminating data transfers, generating the clock which regulates the flow of data, transmitting and receiving data. EEPROMs can only serve as slaves, receiving and transmitting data.

General information about data transfer formats for devices on the I²C-bus will be given now, for more detailed information regarding SLx 24Cxx please refer to the datasheets.

The master initiates a data transfer by generating a start condition (SDA going low while SCL high). The master terminates the data transfer by generating a stop condition (SDA going high while SCL high). Every data put on the SDA line must be 8 bits long. The number of data bytes transferred between the START and STOP condition is not limited by the I²C-bus protocol. The data on the SDA line must be stable during the HIGH period of the clock. Data can only change during the low phase of the clock.

The data byte is transferred serially with the most significant bit first, and followed by an acknowledge bit on every 9th clock pulse (see **figure 1**).

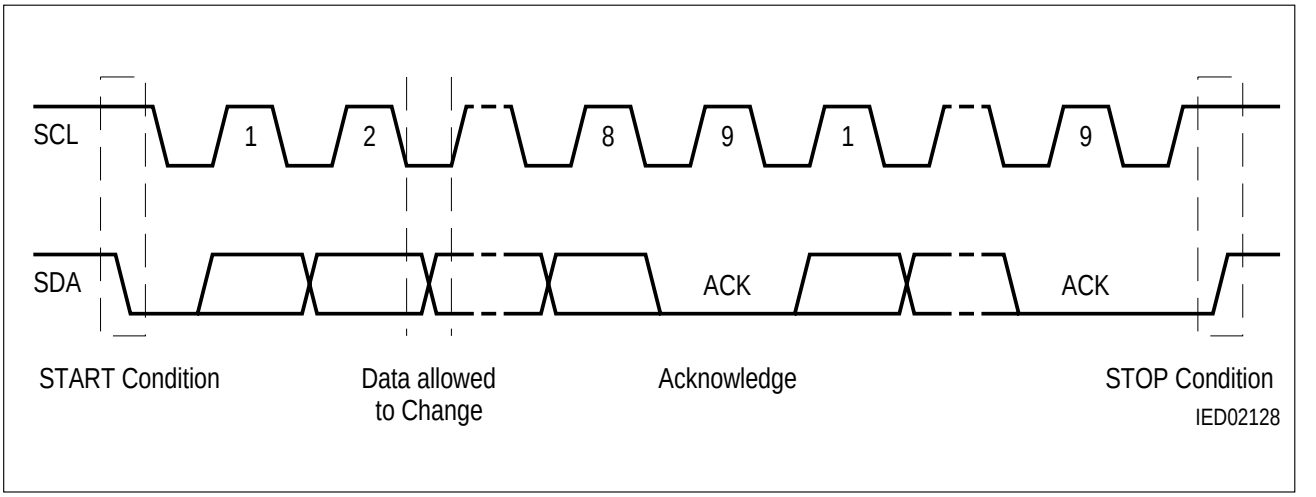

Figure 1
Data Transfer Conventions

The receiver is obliged to generate an acknowledge bit after each byte of data that has been received. If the master is receiving data, it must signal the end of data transfer to the slave by not generating an acknowledge bit on the last byte of data received (see **figure 2**). Then, the slave must release the data line to allow the master to generate the STOP condition.

Figure 3 shows the I²C-bus data transfer format of writing data from master to the slave device. The 8th of the command byte is a data direction bit (R/W). The “0” for data direction bit indicates a transmission (WRITE). A “1” indicates a request for data (READ). **Figure 2** shows the data transfer format of reading data from the slave device.

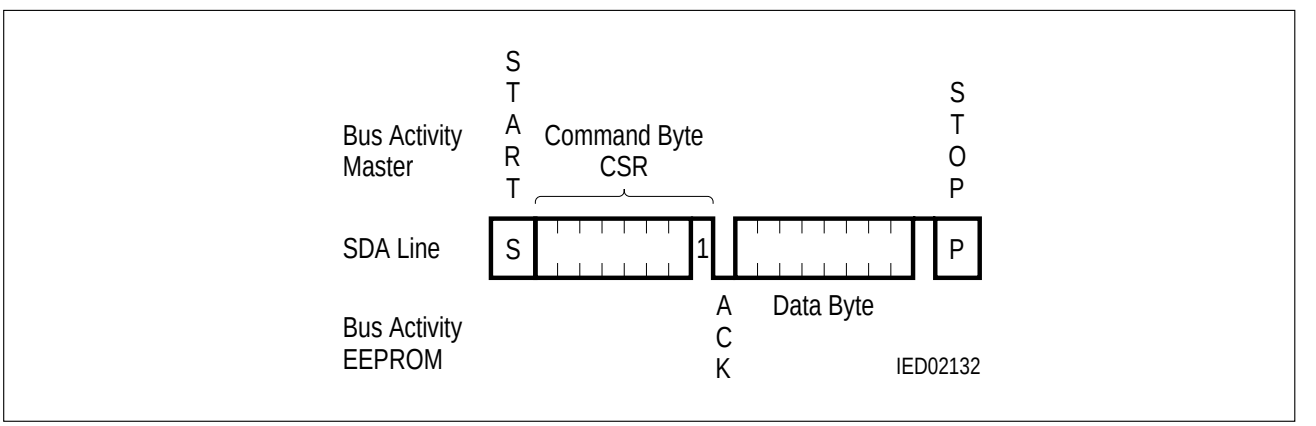

Figure 2
Read Sequence

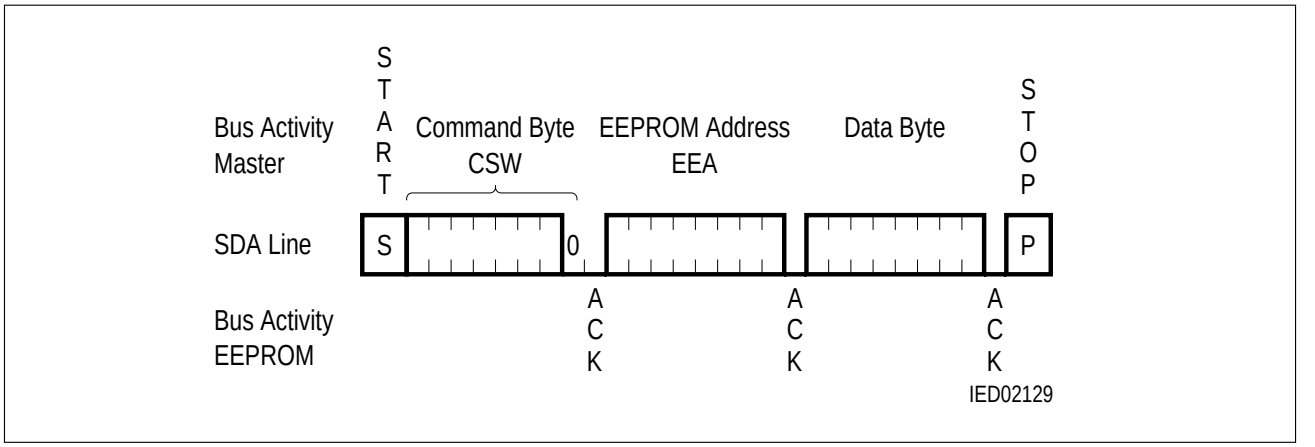

Figure 3
Write Sequence

2.2 Timing Diagram

The clock frequency of SCL is in the range of 0 up to 400 kHz. Occasionally, slave devices may slow down the transmission by holding the clock line low after receiving a byte of data from the microcontroller. This event is defined as a WAIT condition. Therefore, the master needs to switch the SCL output to high impedance and read the SCL line before transmitting another byte of data to the slave device. **Figure 4** shows the data transfer timing requirements in detail. The description of the used abbreviations is shown in the **table 1**. The minimum timing requirements are needed to be fulfilled in order for I²C-bus to operate properly.

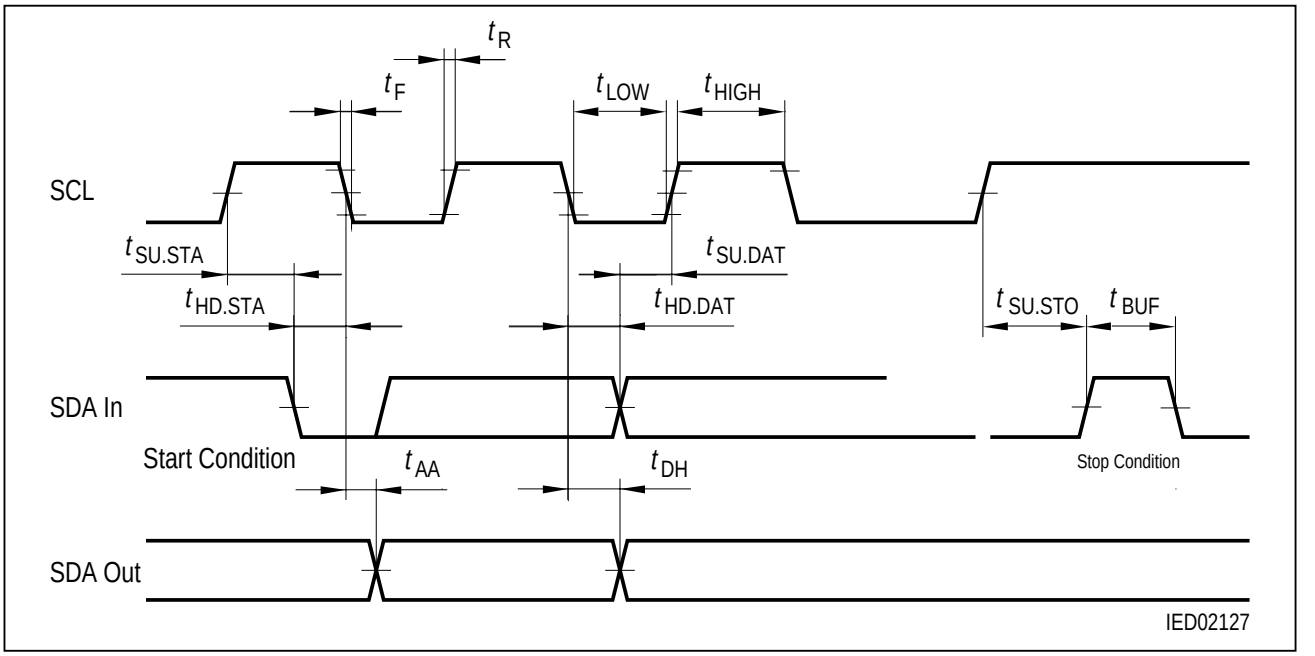

Figure 4
Bus Timing

Table 1

Parameter	Symbol	Standard Mode I²C-bus		Fast Mode I²C-bus		Units
		min.	max.	min.	max.	
SCL clock frequency	f_{SCL}		100		400	kHz
Clock pulse width low	t_{low}	4.7		1.2		µs
Clock pulse width high	t_{high}	4.0		0.6		µs
SDA and SCL rise time	t_R		1000		300	ns
SDA and SCL fall time	t_F		300		300	ns
Start set-up time	$t_{SU.STA}$	4.7		0.6		µs
Start hold time	$t_{HD.STA}$	4.0		0.6		µs
Data in set-up time	$t_{SU.DAT}$	200		100		ns
Data in hold time	$t_{HD.DAT}$	0		0		µs
SCL low to SDA data out valid	t_{AA}	0.1	4.5	0.1	0.9	µs
Data out hold time	t_{DH}	100		50		ns
Stop set-up time	$t_{SU.STO}$	4.0		0.6		µs
Time the bus must be free before a new transmission can start	t_{BUF}	4.7		1.2		µs
Capacitive load for each bus line	C_b		400		400	pF

2.3 Hardware Connection.

Every device connected to the I²C-bus must have an open drain/open collector output for both the clock (SCL) and data (SDA) lines. Each of the lines is connected to the V_{CC} supply via a common pull-up resistor. The connection among master and many slave devices is shown in **figure 5**. The number of devices that can be connected to the I²C-bus is limited only by the maximum bus load capacitance of 400 pF. Theoretically up to 8 serial EEPROMs can be connected to the same I²C-bus. Not all types of the SLx 24Cxx family support this feature, please refer to the datasheets.

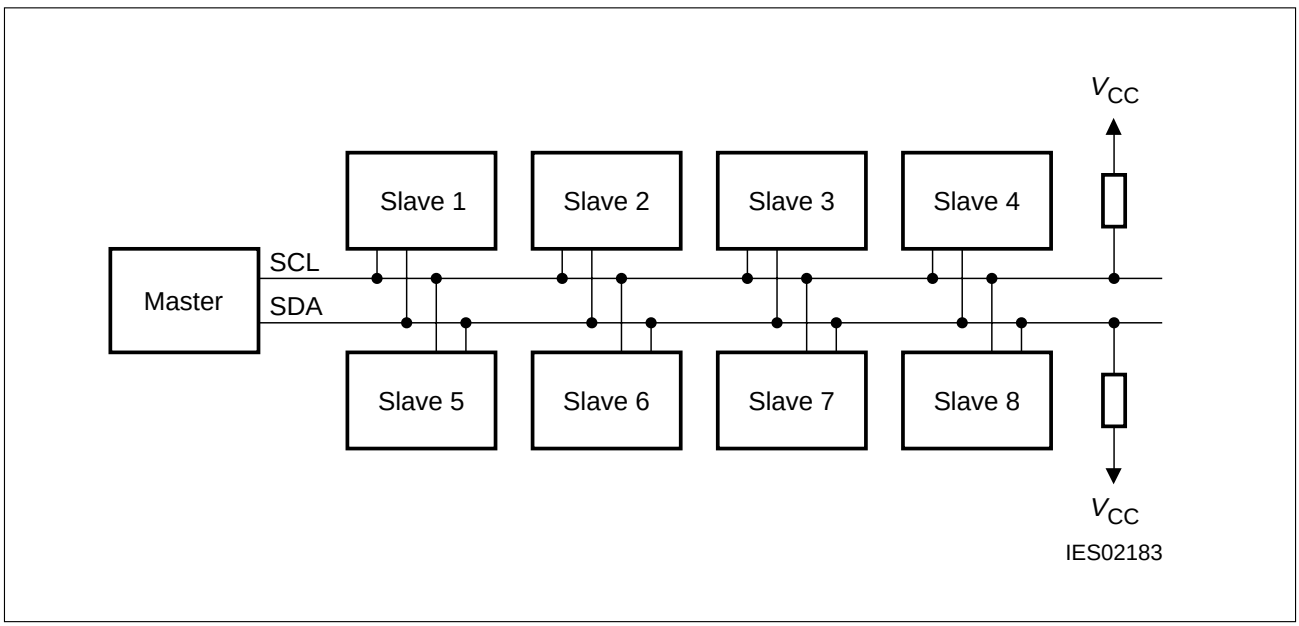

Figure 5
Bus Configuration

3 Software Description

3.1 Software Concept

Table 2

Device Type	Directory	I²C-Bus module	EEPROM module	Demo program
SLx 24C01 SLx 24C02 SLx 24C04 SLx 24C08 SLx 24C16 SLx 24C164	2401_8B	I²C_SW8B.C I²C_SW8B.H I²C_8B.DEF	24SW01.C 24SW01.H	24dem01.C
SLx 24C01/P SLx 24C02/P	2401_8BP	I²C_SW8B.C I²C_SW8B.H I²C_8B.DEF	24SW01P.C 24SW01P.H	24dem01P.C
SLx 24C04/P SLx 24C08/P SLx 24C16/P SLx 24C164/P	2404_8BP	I²C_SW8B.C I²C_SW8B.H I²C_8B.DEF	24SW01P.C 24SW01P.H	24dem04P.C
SLx 24C32 SLx 24C64	2432_8B	I²C_SW8B.C I²C_SW8B.H I²C_8B.DEF	24SW32.C 24SW32.H	24dem32.C
SLx 24C32/P SLx 24C64/P	2432_8BP	I²C_SW8B.C I²C_SW8B.H I²C_8B.DEF	24SW32P.C 24SW32P.H	24dem32P.C

The software for each device type (e.g. Slx 24C01) consists of 1 basic I²C-bus module, 1 control module with EEPROM routines and 1 demo program. The I²C-bus module contains subroutines for the I²C-bus protocol. The EEPROM module uses these subroutines to implement command sequences for SLx 24Cxx devices. The demo program shows how the modules can be used in an application program. **Table 2** gives an overview which modules and demo program have to be used for the different device types and in which directory of the `24Cxx8b.EXE` these files can be found.

In the first module (`I2C_SW8B.C`) the clock and data of the I²C-bus are generated by using CPU time of the 8051 microcontroller. The clock frequency of the I²C-bus is approximately 100 kHz with 20 MHz CPU and 300 ns for a one-cycle instruction. The time delay is realized with NOP commands. If the CPU clock is less or more than 20 MHz or an one-cycle instruction takes more or less than 300 ns, this software module can be modified to achieve the 100 or 400 kHz of SCL frequency.

The software modification involves reducing or eliminating the time delay. In addition, check for WAIT condition before every byte is sent or received rather than checking for WAIT condition before every bit is sent or received.

Be aware of the fact that this software module can be interrupted by others software modules that are involved in hardware interrupts. In other words, this software module is equivalent to those software modules with lowest priority interrupt level.

The advantage of using the CPU time to generate clock and data is that only use 2 port pins and no other hardware peripheral of the 8051 microcontroller are used. Therefore, those hardware peripherals can be reserved for other time critical software application. The `I2C_SW8B.C` software module is divided into 5 software subroutines which can be accessed by an external programs. These 5 software basic subroutines are used to construct the data transfer format of the I²C-bus: `I2cInit`, `I2cStart`, `I2cMasterWrite`, `I2cMasterRead` and `I2cStop`.

The second module (`24SWxx.C`) shows the whole functionality of SLx 24Cxx I²C-bus serial EEPROMs using the basic subroutines of the first module (`I2C_SW8B.C`). All `24SWxx.C` software modules are divided into 7 software subroutines which can be accessed by external programs: `ByteWriteE2prom`, `PageWriteE2prom`, `Ack_pol_r`, `Ack_pol_w`, `RandomReadE2prom`, `CurrentReadE2prom` and `SequentialReadE2prom`. The routines `Ack_pol_r` and `Ack_pol_w` contain time critical wait cycles that are calculated for 20 MHz CPU and 300 ns for a one-cycle instruction. If the CPU clock is less or more than 20 MHz or an one-cycle instruction takes more or less than 300ns, these routines might be modified. For devices with Page Protection Mode™ (SLx 24Cxx.../P) the software modules `24SWxxP.C` contain additionally to the 7 basic subroutines 3 routines especially for the Page Protection Mode™: PPMWriteE2prom, PPMEraseE2prom and PPMReadE2prom.

The third part of this software package contains demo programs for devices without (`24demxx.C`) and with Page Protection Mode™ (`24demxxP.C`). These demo programs demonstrate how the software modules `24SWxx.C/24SWxxP.C` can be used in an application program.

3.2 Description of the Software Routines

I²C-BUS Software Module

Source file: `I2C_SW8B.C`

Header file: `I2C_SW8B.H`

User definition file: `I2C_8B.DEF`

Description

The I²C-bus module simulates a standard I²C-bus single master protocol by using CPU time of the 8051 microcontroller. The clock as well as transmit/receive data are handled by the CPU time delay. The 2 port pins of the microcontroller which are used for clock and transmit/receive data are defined in `I2C_8B.DEF`.

Module Subroutines

1. `void CheckClock();`
2. `unsigned char Check_SCL();`
3. `unsigned char I2cInit();`
4. `void I2cStart();`
5. `unsigned char I2cMasterWrite(unsigned char input_byte);`
6. `unsigned char I2cMasterRead(unsigned char ack);`
7. `unsigned char I2cStop();`

`void CheckClock()`

Send HIGH and read the SCL line. It will wait until the line has been released from slave device (SLx 24Cxx) for every bit of data to be sent or received.

Parameter	Description
None	

`unsigned char Check_SCL()`

Send HIGH and read the SCL line. It will wait until the line has been released from slave device (SLx 24Cxx) with the time-out of approximately 10 ms.

Parameter	Description
None	

Return

The return value is “0” if the clock and data lines have no problem. Otherwise, the return value will be “1”.

`unsigned char I2cInit()`

Check clock and data lines for any bus faulty like no pull-up resistor on SDA/SCL or pull-down to low by the slave device.

Parameter	Description
None	

Return

The return value is “0” if the clock and data lines have no problem. Otherwise, the return value will be “1”.

`void I2cStart()`

Generate a START condition on I²C-bus.

Parameter	Description
None	

`unsigned char I2cMasterWrite(unsigned char input_byte)`

Output one byte of data to the slave device. Check for WAIT condition before every bit is sent.

Parameter	Description
`unsigned char input_byte`	one byte of data to be sent to slave device.

Return

If the return value is “0”, write sequence is received from slave device (SLx 24Cxx) sucessfully. Otherwise, write sequence is needed to be sent again because there is no acknowledge from the slave device.

`unsigned char I2cMasterRead(unsigned char ack)`

Read one byte of data from the slave device. Check for WAIT condition before every bit is received.

Parameter	Description
`unsigned char ack`	“0” - generate LOW output by the master after a byte of data is received. “1” - generate HIGH output by the master after a byte of data is received.

Return

If the return value is “0”, reading one byte of data from slave device (SLx 24Cxx) is successfully. Otherwise, one byte of data is needed to be received again, because there is no acknowledge from the slave device.

`unsigned char I2cStop()`

Generate a STOP condition on the I²C-bus. In addition, it will generate clock pulses until the line is released by the slave device (SLx 24Cxx) and the time-out is approximately 10 ms before returning a “HIGH” value indicating an error.

Parameter	Description
None	

Return

The return value is “0” if the clock and data lines have no problem. Otherwise, the return value will be “1”.

I²C-Bus EEPROM Software

Source file: `24SW01P.C`

Header file: `24SW01P.H`

Description

The EEPROM module `24SW01P.C` shows the whole functionality of SLx 24Cxx EEPROMs and contains additionally 3 subroutines for Page Protection Mode™ using the I²C-bus module (`I2C_SW8B.C`). **Table** gives a short overview about the main differences between the module `24SW01P.C` and the other EEPROM modules.

Table 3

EEPROM modules	Main differences to module `24SW01P.C`
`24SW01.C`	only subroutines 1-7 are included
`24SW032.C`	only subroutines 1-7 are included, instead of 1 address byte (`unsigned char address`) for addressing, 2 address bytes are used (`unsigned char addresshi, unsigned addresslo`)
`24SW32P.C`	instead of 1 address byte (`unsigned char address`) for addressing, 2 address bytes are used (`unsigned char addresshi, unsigned addresslo`)

Software routines

```
1. unsigned char ByteWriteE2prom(unsigned char command,
     unsigned char address,unsigned char *buffer);
2. unsigned char PageWriteE2prom (unsigned char command,
     unsigned char address, unsigned char *buffer,
     unsigned char count);
3. unsigned char Ack_pol_r(unsigned char command);
4. unsigned char Ack_pol_w(unsigned char command);
5. unsigned char RandomReadE2prom (unsigned char command,
     unsigned char address, unsigned char *buffer);
6. unsigned char CurrentReadE2prom (unsigned char command,
     unsigned char *buffer);
7. unsigned char SequentialReadE2prom(unsigned char command,
     unsigned char address, unsigned char *buffer, unsigned char count);
```

These routines are only for SLx 24Cxx/P with Page Protection Mode™ :

```
8. unsigned char PPMWriteE2prom(unsigned char command,
     unsigned char address, unsigned char *buffer,
     unsigned char count);
9. unsigned char PPMEraseE2prom(unsigned char command,
     unsigned char address, unsigned char *buffer,
     unsigned char count);
10.unsigned char PPMReadE2prom(unsigned char command,
     unsigned char address, unsigned char *buffer,
     unsigned char count);
```

```
unsigned char ByteWriteE2prom(unsigned char command,
 unsigned char address, unsigned char *buffer)
```

Write 1 data byte to E²PROM. The flow of this subroutine is derived from the data format of writing as shown in **figure 3**.

Parameter	Description
None	

Return

If the return value is “0”, the programming of E²PROM is started. Otherwise, the data is needed to be sent again because there is no acknowledge from slave device.

```
unsigned char PageWriteE2prom(unsigned char command,
 unsigned char address,unsigned char *buffer,unsigned char count)
```

Write number of data bytes to E²PROM.

Parameter	Description
`unsigned char command`	specifies the command byte
`unsigned char address`	specifies the EEPROM address
`unsigned char *buffer`	point to the location of data to be sent
`unsigned char count`	number of bytes to be sent

Return

If the return value is “0”, the programming of E²PROM is started. Otherwise, the data is needed to be sent again because there is no acknowledge from slave device.

`unsigned char Ack_pol_r(unsigned char command)`

Check for the completion of programming by sending read commands. If the programming is completed, the acknowledge bit will be “0”.

Parameter	Description
`unsigned char command`	specifies the command byte

Return

If the return value is “0”, the programming of E²PROM is considered to be done. Otherwise, the programming last much more than 8 ms (CPU clock 20 MHz and 200 ns for a one-cycle instruction) and an error occurred.

`unsigned char Ack_pol_w(unsigned char command);`

Check for the completion of programming by sending write commands. If the programming is completed, the acknowledge bit will be “0”.

Parameter	Description
`unsigned char command`	specifies the command byte

Return

If the return value is “0”, the programming of E²PROM is considered to be done. Otherwise, the programming last much more than 8 ms (CPU clock 20 MHz and 200 ns for a one-cycle instruction) and an error occurred.

```
unsigned char RandomReadE2prom(unsigned char command,
 unsigned char address,unsigned char *buffer)
```

Read 1 data byte from E²PROM.

Parameter	Description
`unsigned char command`	specifies the command byte
`unsigned char address`	specifies the EEPROM address
`unsigned char *buffer`	point to the location of data to be stored

Return

If the return value is “0”, the reading of E²PROM is completed. Otherwise, the data is needed to be sent again, because there is no acknowledge from slave device.

```
unsigned char CurrentReadE2prom (unsigned char command,
 unsigned char *buffer)
```

Read 1 data byte from the current E²PROM address. The flow of this subroutine is derived from the data format of reading from the E²PROM as in the **Figure 3**.

Parameter	Description
`unsigned char command`	specifies the command byte
`unsigned char *buffer`	point to the location of data to be stored

Return

If the return value is “0”, the reading of E²PROM is completed. Otherwise, the data is needed to be sent again, because there is no acknowledge from slave device.

```
unsigned char SequentialReadE2prom (unsigned char command,
 unsigned char address,unsigned char *buffer,
 unsigned char count)
```

Read number of data bytes from the E²PROM.

Parameter	Description
`unsigned char command`	specifies the command byte
`unsigned char address`	specifies the EEPROM address
`unsigned char *buffer`	point to the location of data to be sent
`unsigned char count`	number of bytes to be sent

Return

If the return value is “0”, the reading of E²PROM is completed. Otherwise, the data is needed to be sent again, because there is no acknowledge from slave device.

```
unsigned char PPMWriteE2prom(unsigned char command,
 unsigned char address, unsigned char *buffer,
 unsigned char count)
```

Protect one page. Therefore the PPM bit is set to “0”.

Parameter	Description
`unsigned char command`	specifies the command byte
`unsigned char address`	specifies the EEPROM address
`unsigned char *buffer`	point to the location of data to be sent
`unsigned char count`	byte number of one page to be sent

Return

If the return value is “0”, the programming of the PPM bit is started. Otherwise, the data is needed to be sent again because there is no acknowledge from slave device.

```
unsigned char PPMEraseE2prom(unsigned char command, unsigned char
             address,unsigned char *buffer,unsigned char count)
```

Unprotects one page. Therefore the PPM bit is set to "1".

Parameter	Description
`unsigned char command`	specifies the command byte
`unsigned char address`	specifies the EEPROM address
`unsigned char *buffer`	point to the location of data to be sent
`unsigned char count`	byte number of one page to be sent

Return

If the return value is “0”, the programming of the PPM bit is started. Otherwise, the data is needed to be sent again because there is no acknowledge from slave device.

```
unsigned char PPMReadE2prom(unsigned char command, unsigned char
            address,unsigned char *buffer, unsigned char count)
```

Read status of PPM bits from the E²PROM.

Parameter	Description
`unsigned char command`	specifies the command byte
`unsigned char address`	specifies the EEPROM address
`unsigned char *buffer`	point to the location of data to be stored
`unsigned char count`	number of PPM bits to be read

Return

If the return value is “0”, the reading of E²PROM is completed. Otherwise, the data is needed to be sent again because there is no acknowledge from slave device.

I²C-BUS EEPROM Demonstration Program

Description

The demo program shows how the `24SWxx.C/24SWxxP.C` software modules can be used in an application program. To verify whether all routines of the demo program runs successfully, **table** and show the memory content before/after running the demo program.

Table 4

	address	Initial memory content	Memory content after running demo program
`24dem01.C`	000 008	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	07 01 02 03 04 05 03 07 07 01 02 03 04 05 03 07
`24dem32.C`	000 008	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	07 01 02 03 04 05 03 07 07 01 02 03 04 05 03 07

Table 5

	address	[1]	Initial memory content	[1]	Memory content after running demo program
`24dem01P.C`	000 008 	U U U	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	P U U	07 01 02 03 04 05 03 07 07 01 02 03 04 05 03 07 0[2] 0[2] 1[2]FF FF FF FF FF
`24dem04P.C`	000 008 010 018 020	U - U - U	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	P - U - U	07 01 02 03 04 05 03 07 07 01 02 03 04 05 03 07 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 0[2] 0[2] 1[2] FF FF FF FF FF
`24dem32P.C`	000 008 010 018 020 028 030 038 040	U - - - U - - - U	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	P - - - U - - - U	07 01 02 03 04 05 03 07 07 01 02 03 04 05 03 07 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 0[2] 0[2] 1[2] FF FF FF FF FF

[1] U = protection bit “1”, page is unprotected; P = protection bit “0”, page is protected

[2] Only the MSB bit is specified to “0” or “1”(e.g. “0” = 7F or 08, …; e.g. “1” =8F or A5, …)

3.3 Software Compilation

The compilation of this software is using the KEIL C8051 compiler. First of all, open µVision/51 from Keil. Under the PROJECT menu click on “New Project” and enter the project name. Then add the I²C-bus module, an EEPROM module and a demo program to the project according to **table 2** (e.g. `I2C_SW8B.C`, `24sw01P.C` and `24dem01P.C`) and save the project. To complete the compilation also the header and definition files are necessary (e.g. `I2C_SW8.H`, `24sw01P.H`,`I2C_8B.DEF`)

Now the project is ready to compile and link all the object files. The compiling and linking of the project can be done by clicking the icon “BUILD ALL”.

The `24Cxx8b.EXE` is a compressed file containing 5 directories. Each directory contains all header, definition and C files which are needed to run the demo program for a special device type (see **table 2**).