SIEMENS

Microcontrollers

ApNote AP1626

☑ additional file
AP162601.EXE available

Software emulation of the I^2C-bus using the High-Speed Synchronous Serial Interface of C166 family

This is a software emulation of I^2C-bus using High-Speed Synchronous Serial Interface of the C166 family to generate the clock and data. The I^2C-bus is used in many applications mainly to communicate between devices connected to the bus.

Author: Tan Choon Hock / SCPL HL RM LAB

AP1626 ApNote - Revision History		
Actual Revision : Rel. 01	Previous Revision: none (Original Version)	
Page of actual Rel.	Page of prev. Rel.	

1 Introduction to I^{2}C-bus

The I^{2}C-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-wire bus. The two bus lines are serial clock line (SCL), and serial data line (SDA). Nowadays, the I^{2}C-bus becomes a standard bus system which is used in consumer electronics, telecommunications, and industrial electronics. This software module of I^{2}C-bus emulation supports the single master protocol only. It is using the High-Speed Synchronous Serial Interface to generate clock, and transmit or receive the data. The clock frequency of the I^{2}C-bus can achieve up to 100 KHz with 20 MHz CPU of the C16x microcontroller.

2 I^{2}C-bus Specifications

2.1 Data Transfer formats

A HIGH-to-LOW transition of the data line (SDA) while the clock line (SCL) is HIGH indicates a START condition. A LOW-to-HIGH transition of the SDA while SCL is HIGH defines a STOP condition. The data line can only be changed when the clock signal on the SCL line is LOW. Therefore, the data on the SDA line must be stable during the HIGH period of the clock signal. The bus is considered to be busy after the Start condition and is considered to be free at a certain time interval after the STOP condition.

Each information puts on the SDA line must be 8-bit long. The data is transferred serially with the most significant bit first, and followed by an acknowledge bit. The 9th clock pulse of the acknowledge bit is generated by the master. The transmitting device has to release the SDA line (HIGH or in the high impedance state) during this clock pulse while the device that needs to acknowledge has to pull down the SDA line during this clock pulse. The number of data bytes transferred between the START and STOP condition from the transmitter and receiver is not limited.

The receiver is obliged to generate an acknowledge bit after each byte of data that has been received. When the receiver does not provide an acknowledge bit after having received a byte of data, the data line must be left HIGH or in the high impedance state by the slave. The master can then generate a STOP condition to abort the transfer. One of the reasons for the receiver not providing the acknowledge bit is that the receiver is performing some real-time function. If the master is receiving data, it must signal the end of the data to the slave by not generating an acknowledge bit on the last byte of data received. Then, the slave must release the data line to allow the master to generate the STOP condition.

A complete data transfer format is shown in Figure 1. After a START condition, a slave address is sent. The address is 7 bits long followed by an 8th bit which is a data direction bit (R/W). A “0” for data direction bit indicates a transmission (WRITE), and a “1” indicates a request for data (READ). Figure 2 shows the I^{2}C-bus data transfer format of writing data from master to slave device. Figure 3 shows the data transfer format of reading data from the slave device.

A data transfer is always terminated by STOP condition generated by the master. However, if the master still wishes to communicate on the bus, it can generate a repeated START condition and address the same device or another slave device without first generating a STOP condition. This combined data transfer format is shown in figure 4.

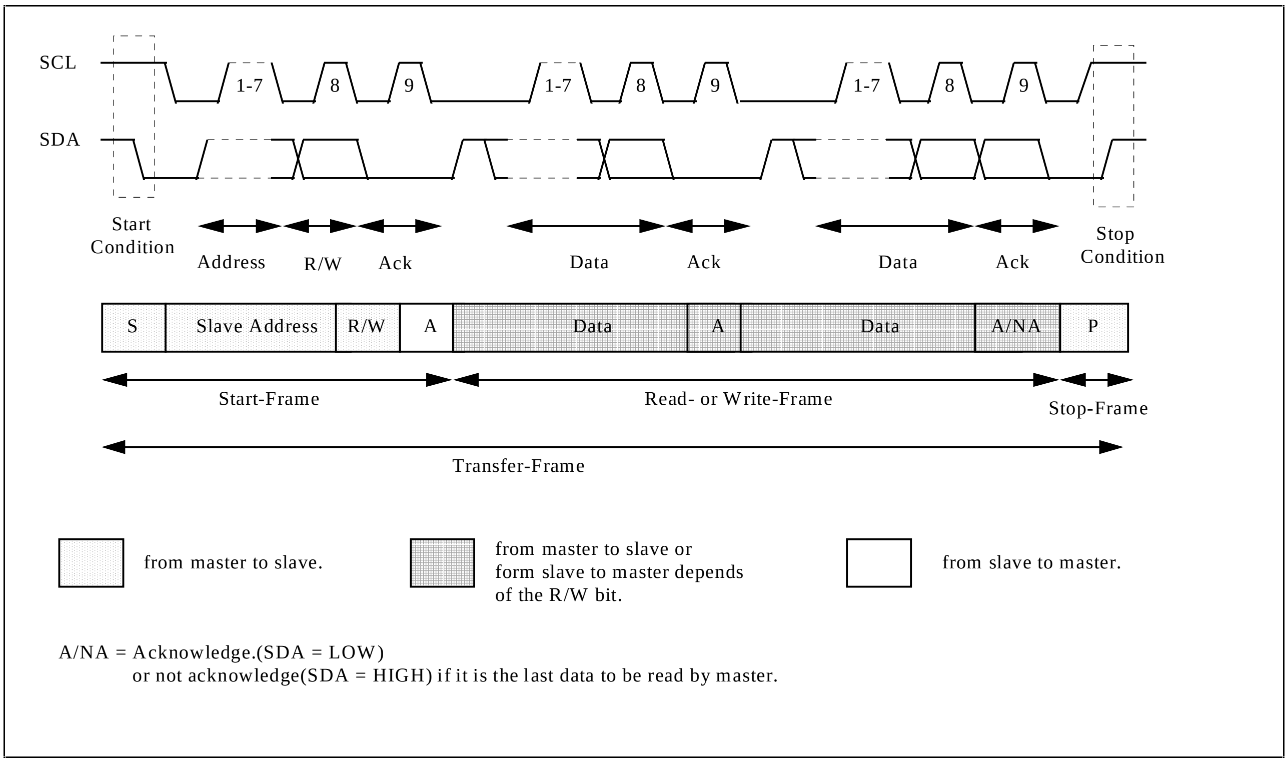

Figure 1:

A complete data transfer format of I^{2}C-bus

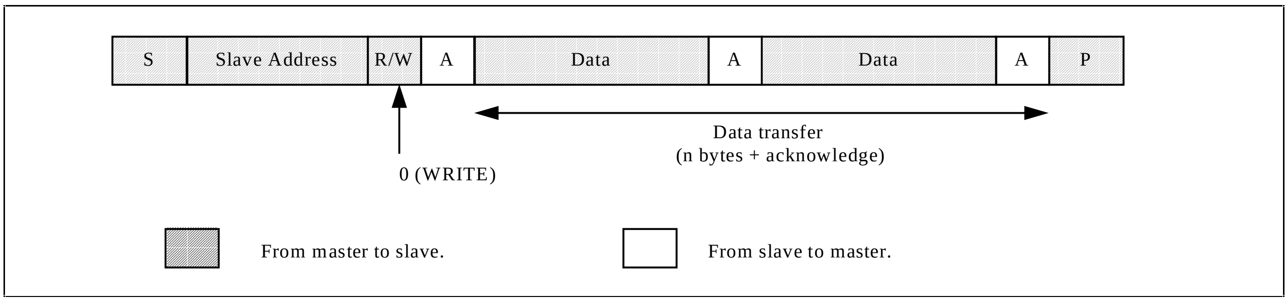

Figure 2:

I^{2}C-bus data transfer format of writing data to slave

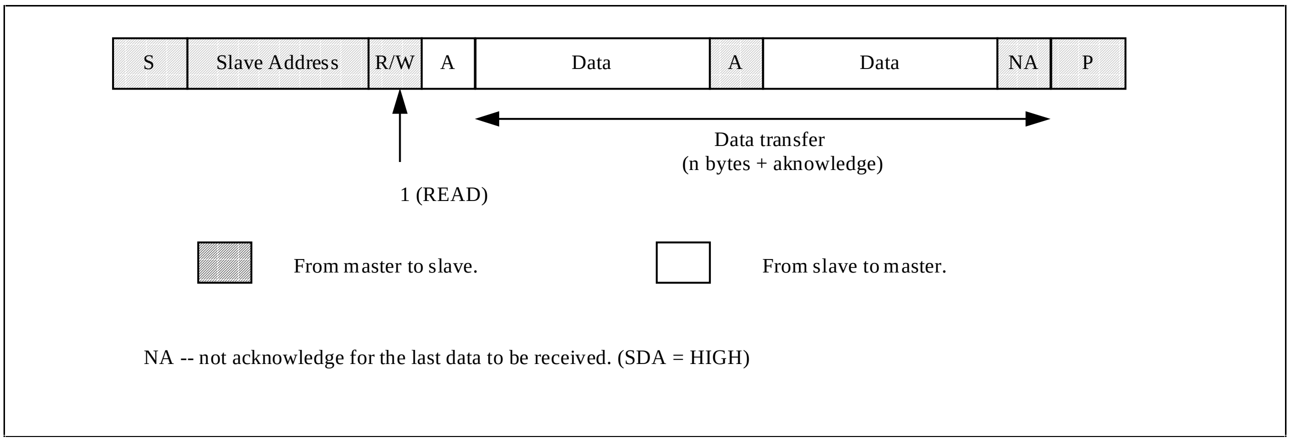

Figure 3:

I^{2}C-bus data transfer format of reading data from slave

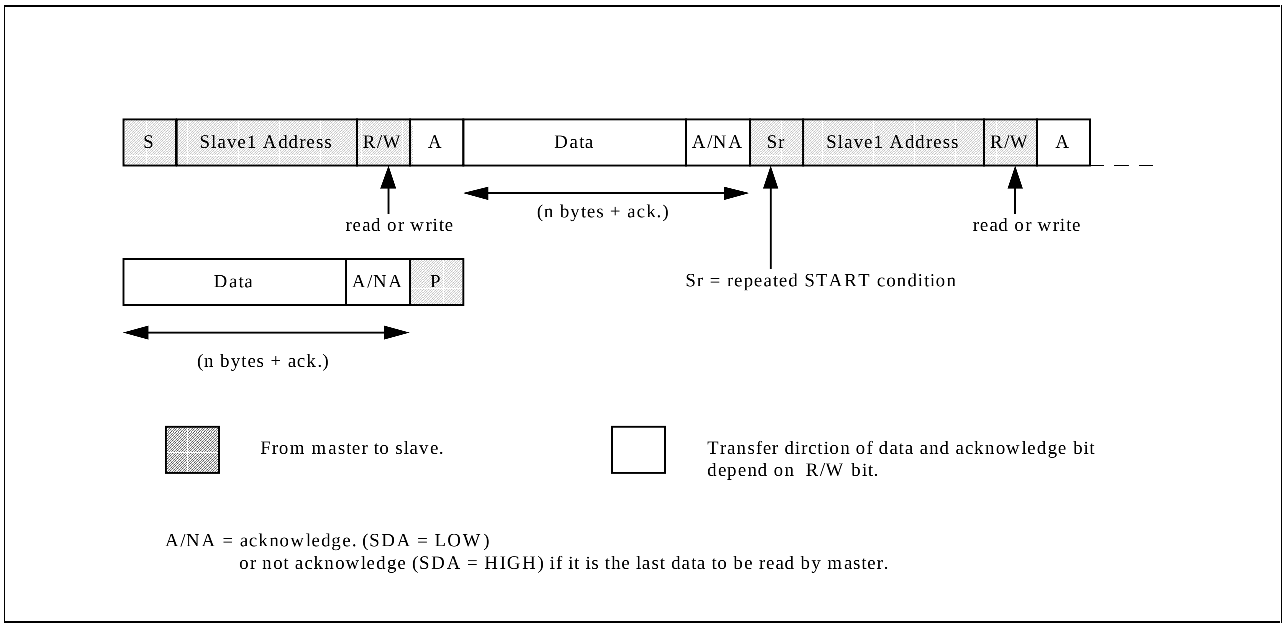
S
Slave1 Address
R/W
A
Data
A/NA
Sr
Slave1 Address
R/W
A
read or write
(n bytes + ack.)
read or write
Data
A/NA
P
Sr = repeated START condition
(n bytes + ack.)
From master to slave.
Transfer dirction of data and acknowledge bit depend on R/W bit.
A/NA = acknowledge. (SDA = LOW)
or not acknowledge (SDA = HIGH) if it is the last data to be read by master.

Figure 4:

A combined data transfer format for I^{2}C-bus

2.2 Timing Diagram

The clock frequency of SCL is in the range of 0 up to 100 KHz. The clock on the I^{2}C-bus has a minimum LOW period of 4.7 µs , and a minimum HIGH period of 4.0 µs.

Occasionally, the slave device may slow down the transmission by holding the clock line low after receiving a byte of data from 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 5 shows the data transfer timing requirements in detail. The description of the abbreviations used is shown in the Table 1. The minimum timing requirements are needed to be fulfilled in order for I^{2}C-bus to operate properly.

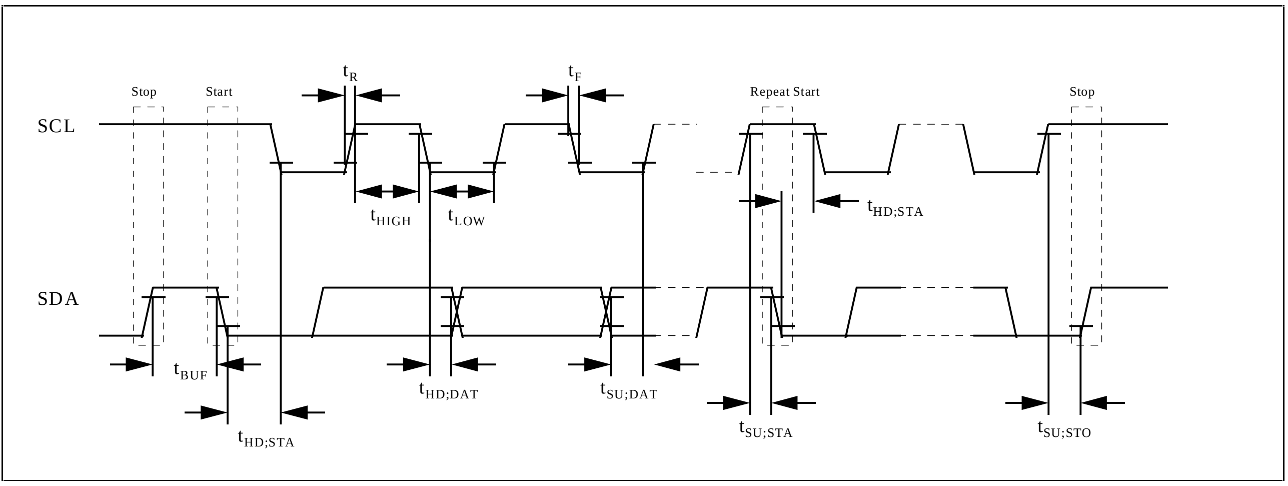

Figure 5:

I^{2}C-bus timing diagram

Table 1:

Abbreviation for I^{2}C-bus timing diagram

Parameter	**Symbol**	**Limit Values**		**Unit**
		min.	**max.**	
1. Bus free time between a STOP and START condition	t_{BUF}	4.7		µs
2. Hold time for START condition. After this period, the first pulse is generated.	$t_{HD;STA}$	4.0		µs
3. The HIGH period of SCL clock.	t_{HIGH}	4.0		µs
4. The LOW period of SCL clock.	t_{LOW}	4.7		µs
5. Data hold time	$t_{HD;DAT}$	0*****		µs
6. Rise time for both SCL and SDA signals.	t_R		1.0	µs
7. Fall time for both SCL and SDA signals.	t_F		300	ns
8. Data set-up time	$t_{SU;DAT}$	250		ns
9. Set-up time for a repeated START condition.	$t_{SU;STA}$	4.7		µs
10. Set-up time for STOP condition.	$t_{SU;STO}$	4.0		µs
11. SCL clcok frequency	f_{SCL}	0	100	KHz
12. Capacitor load for each bus line	C_b		400	pF

***** A device must internally provide a hold time of at least 300 ns for SDA signal in order to bridge the undefined region of the falling edge of SCL.

2.3 Hardware Connection

Every device connected to the I^2C-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 VDD supply via a common pull-up resistor of 10 KΩ in value. The connection among master and many slave devices is shown in figure 6. The number of devices that can be connected to the I^2C-bus is limited only by the maximum bus load capacitance of 400 pF.

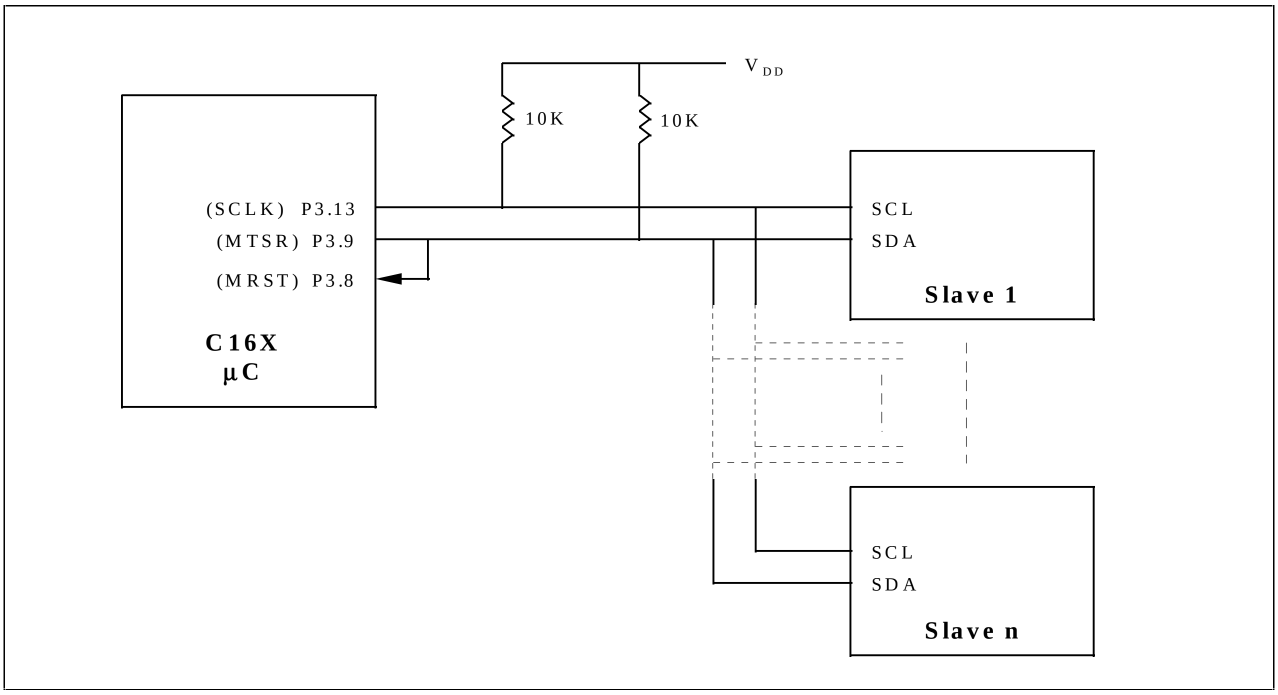

Figure 6:

Hardware connection among master and slave devices

3 Software Description

3.1 Software Concept

The clock and data of the I^{2}C-bus are generated by the High-Speed Synchronous Serial Interface of the C16x microcontroller. The clock frequency of the I^{2}C-bus is 100 KHz with 20 MHz CPU of the microcontroller. The baud rate generator for the SCL line can be determined by the following formula:

$$B_{SSC} = \frac{f_{CPU}}{2 * (<SSCBR> + 1)}$$

$$SSCBR = \left(\frac{f_{CPU}}{2 * Baudrate_{SSC}} \right) - 1$$

<SSCBR> represents the content of the reload register.

For example, SSCBR = (20 MHz/(2*100 KHz)) -1 = 99 or 63H. In order to achieve 100 KHz of SCL clock frequency, SSCBR must be assigned as 63H which is based on 20 MHz CPU clock of the 16x microcontroller.

The HS-SSC uses 3 I/O lines for communication; P3.13 (SCLK) serves as the clock line, P3.8 (MRST) serves as the serial data input line (master receive/slave transmit), and P3.9 (MTSR) serves as the serial data output (master transmit/slave receive). The P3.8 and P3.9 pins are connected together as the SDA line for half duplex operation.

The operating mode of the SSC is controlled by the control register SSCCON which has been configured as the 8 bit data with MSB first, in master mode, shift transmit data on the leading clock edge, latch on trailing edge, idle clock line is low, and leading clock edge is low-to-high transition.

This software module does not check the clock line before a byte of data is sent or received. This is because the clock line is controlled by the hardware peripheral of SSC. In other words, the WAIT condition will not be detected before a byte of data is sent or received. Therefore, it is compulsory to check the acknowledge bit. If the acknowledge bit is not received from the slave device, the data is needed to transmit again. The WAIT condition can only be checked before START and after STOP condition in this case.

The `I2C_SW.C` software module is divided into 5 software subroutines which can be accessed by the main or external program. Those 5 software subroutines are used to construct the data transfer format of the I^{2}C-bus. Those 5 software subroutines are `I2cInit`, `I2cStart`, `I2cMasterWrite`, `I2cMasterRead`, and `I2cStop`.

The two types of data transfer format (master write, and master read/combined format) are written in the `HSSCTEST.C`. The `HSSCTEST.C` is a simple test program which just to verify the `I2C_HSSC.C` software module. This test program is to transmit 10 bytes of data to E^{2}PROM IC from the array location of the microcontroller. The 10 bytes of data will be stored in the word address 0 to 9 of the E^{2}PROM IC. Next, the microcontroller will read back the contents of the 10 bytes from the word address 0 to 9 of the E^{2}PROM IC and then store it into another array location of the microcontroller.

3.2 Description of Module Subroutines

I^{2}C-BUS Software Module

Source file:	`I2C_HSSC.C`
Header file:	

Description

This module is a standard I^{2}C-bus single master protocol by using High-Speed Synchronous Serial Interface of the C16x microcontroller. The clock and as well as transmit/receive data are handled by the hardware peripheral of HS-SSC.

Module Subroutines

1. `void Delay(unsigned int count);`
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 Delay(unsigned int count)`

To generate START and STOP condition, and 9th clock pulse.

Parameter	Description
`unsigned int count`	number of count for time delay.

`unsigned char Check_SCL()`

Send HIGH and read the SCL line. It will wait until the line has been released from slave device with the time-out of 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()`

Initialize the I^{2}C-bus. P3.8 = MRST, P3.9 = MTSR, P3.13 = SCL. The MRST and MTSR are connected together for I^{2}C operation. Configure the High-Speed Synchronous Serial interface to generate 100 Kbaud clock frequency and set the control register to produce a 8 bit data with MSB first, in master mode, shift transmit data on the leading clock edge, latch on trailing edge, idle clock line is low, and leading clock edge is low-to-high transition. It also check the 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 I^{2}C-bus.

Parameter	Description
None	

`unsigned char I2cMasterWrite(unsigned char input_byte)`

Output one byte of data to the slave device.

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

Return
If the return value is “0”, writing one byte of data to slave device is successfully. Otherwise, one byte of data 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.

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”, reaing one byte of data from slave device 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 I^{2}C-bus. In addition, it will generate clock pulses until the line is released by the slave device and the time-out is 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”.

I^{2}C-BUS Application Software

Source file:	`HSSCTEST.C`
Header file:	`I2C_HSSC.H`

Description

This program is just for testing only. The purpose of this program is to make use of the standard I^{2}C protocol module (`I2C_HSSC.C`) to control the nonvolatile memory (E^{2}PROM). This program writes 10 bytes of data into E^{2}PROM in sequence and the data is retrieved from the array location of microcontroller. Then read back 10 bytes of data from the E^{2}PROM at the same location to which they have been programmed and store them in the array location of microcontroller.

Software subroutines

1. `unsigned char CheckWrite();`
2. `unsigned char WriteE2prom(unsigned char address, signed int sub_addr,unsigned char *buffer,unsigned char count);`
3. `unsigned char ReadE2prom (unsigned char address,signed int sub_addr,unsigned char *buffer,unsigned char count);`

`unsigned char CheckWrite()`

Check for the completion of programming after the memory write. If the programming is completed, the acknowledge bit will be "0".

Parameter	Description
None	

Return
If the return value is “0”, the programming of E^{2}PROM is consider done. Otherwise, the programming of E^{2}PROM is still in progress.

`unsigned char WriteE2prom(unsigned char adress, signed int sub_addr,unsigned char *buffer,unsigned char count)`

Write number of data bytes to E^{2}PROM. The flow of this subroutine is derived from the data format of writing to the E^{2}PROM as in the figure 2.

Parameter	Description
`unsigned char address`	specifies the slave device address
`signed int sub_addr`	specifies the sub-address/word 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 E^{2}PROM is completed. Otherwise, the data is needed to be sent again because there is no acknowledge from slave device.

`unsigned char ReadE2prom(unsigned char address, signed int sub_addr,unsigned char *buffer,unsigned char count)`

Read number of data bytes from E^{2}PROM. The flow of this subroutine is derived from the data format of reading from the E^{2}PROM as in the figure 3.

Parameter	Description
`unsigned char address`	specifies the slave device address
`signed int sub_addr`	specifies the sub-address
`unsigned char *buffer`	point to the location of data to be stored
`unsigned char count`	number of bytes to be received

Return
If the return value is “0”, the reading of E^{2}PROM is completed. Otherwise, the data is needed to be sent again because there is no acknowledge from slave device.

3.3 Software Compilation

The compilation of this software is using the KEIL C166 compiler. Firsts of all, under the PROJECT menu, clicks on the “New Project”, then key in the name of this project and add files to the project which are the `I2C_HSSC.C` and `HSSCTEST.C`. Then, save the project. After-that, go to the OPTIONS menu and click on the “C166 Compiler...” . Lastly, select the option under OBJECT and cross the box under “Enable 80C167 instructions”. This option will allow you to use the C16x derivatives. Now the project is really 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 AP162601.EXE is a compressed file and contains `I2C_HSSC.H`, `I2C_HSSC.C`, and `HSSCTEST.C`. All these files are necessary to complete the compilation of the software program.