SIEMENS

Microcontrollers

ApNote AP164510

☑ additional file
AP164510.EXE available

Hardware I^{2}C-bus in slave mode by using polling and interrupt methods for C161RI microcontroller

This is a software module for hardware I^{2}C-bus in slave mode by using software polling and hardware interrupt methods for C161RI microcontroller. The I^{2}C-bus is used in many applications mainly to communicate between devices connected to the bus.

Author: Tan Choon Hock / SCPL HL SAC AP

1 Introduction to I^2C-bus 3

2 I^2C-bus Specifications 3

2.1 Data Transfer Formats 3
2.2 Timing Diagram 7
2.3 Hardware Connection 9

3 Software Description 10

3.1 Software Concept 10
3.1.1 Software polling method 10
3.1.2 Hardware interrupt method 11
3.2 Description of Module Subroutines 12
3.2.1 Software polling method 12
3.2.2 Hardware interrupt method 13

4 Software Listing 14

4.1 Software polling method (SLAVE4.C) 14
4.2 Hardware interrupt method (SLAVE2.C) 17

AP164510 ApNote - Revision History		
Actual Revision : Rel. 1.0 Previous Revison: none		
Page of actual Rel.	Page of prev. Rel.	

1 Introduction to I^2C-bus

The I^2C-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^2C-bus becomes a standard bus system which is used in consumer electronics, telecommunications, and industrial electronics.

This software module can support the multi-master operation for hardware I^2C-bus in slave mode. It is using the internal hardware peripheral of I^2C-bus to receive clock, and transmit or receive the data. The slave address can be configured to 7-bit addressing as well as 10-bit addressing. It can support the clock frequency of the I^2C -bus up to 400 KHz with 16 MHz CPU of the C161RI microcontroller.

2 I^2C-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 reason for the receiver not to provide 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^2C-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.

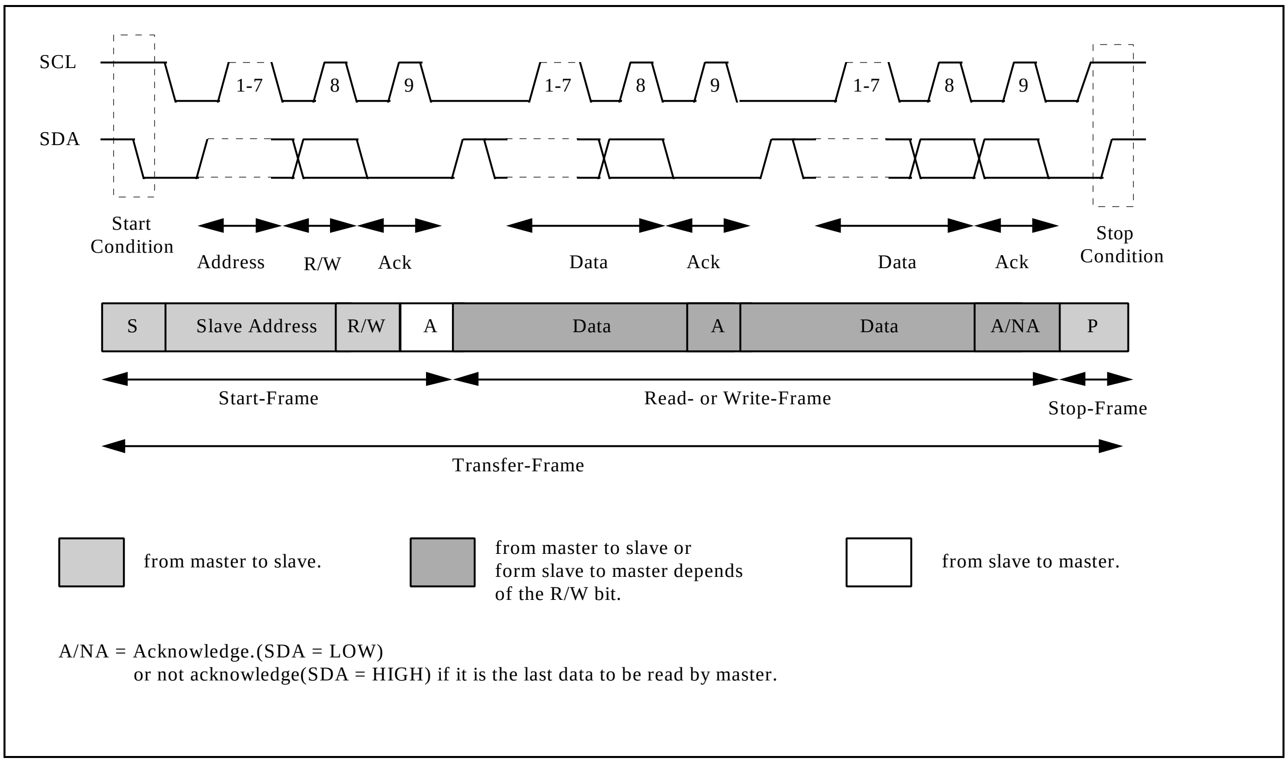

Figure 1:

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

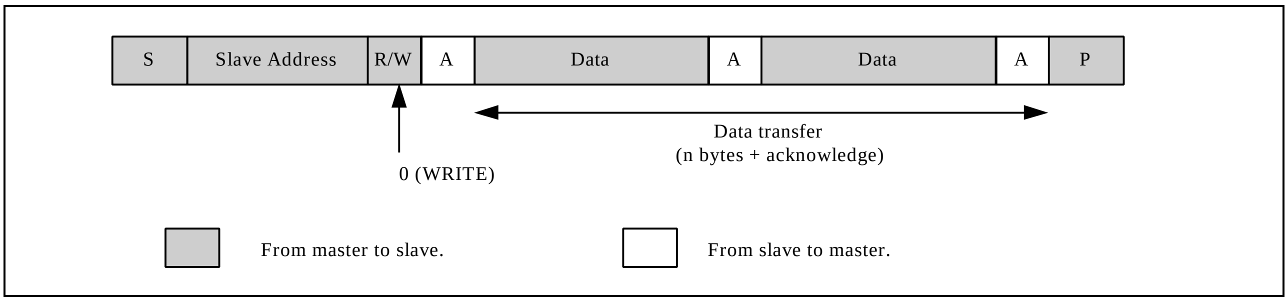

Figure 2:

I²C-bus data transfer format of writing data to slave

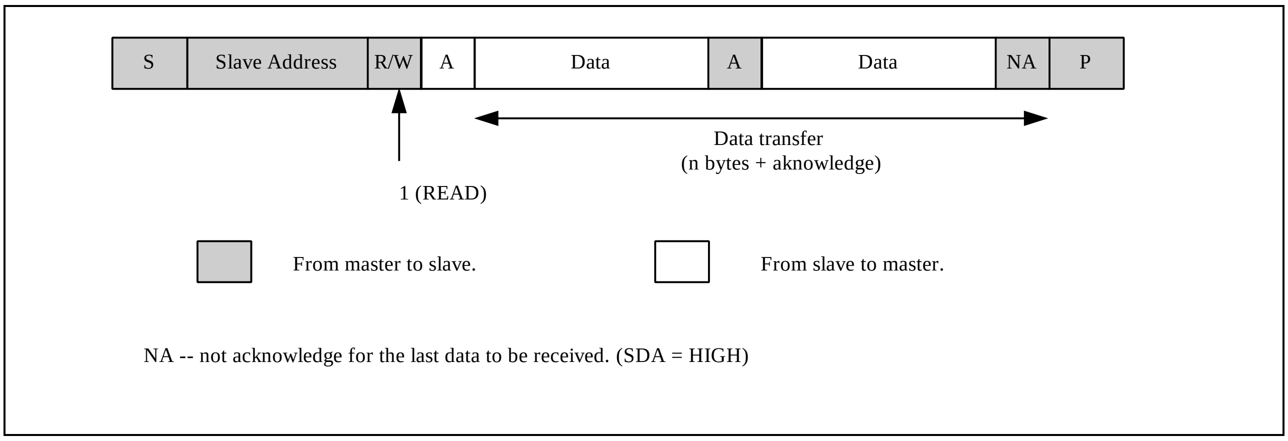

Figure 3:

I²C-bus data transfer format of reading data from slave

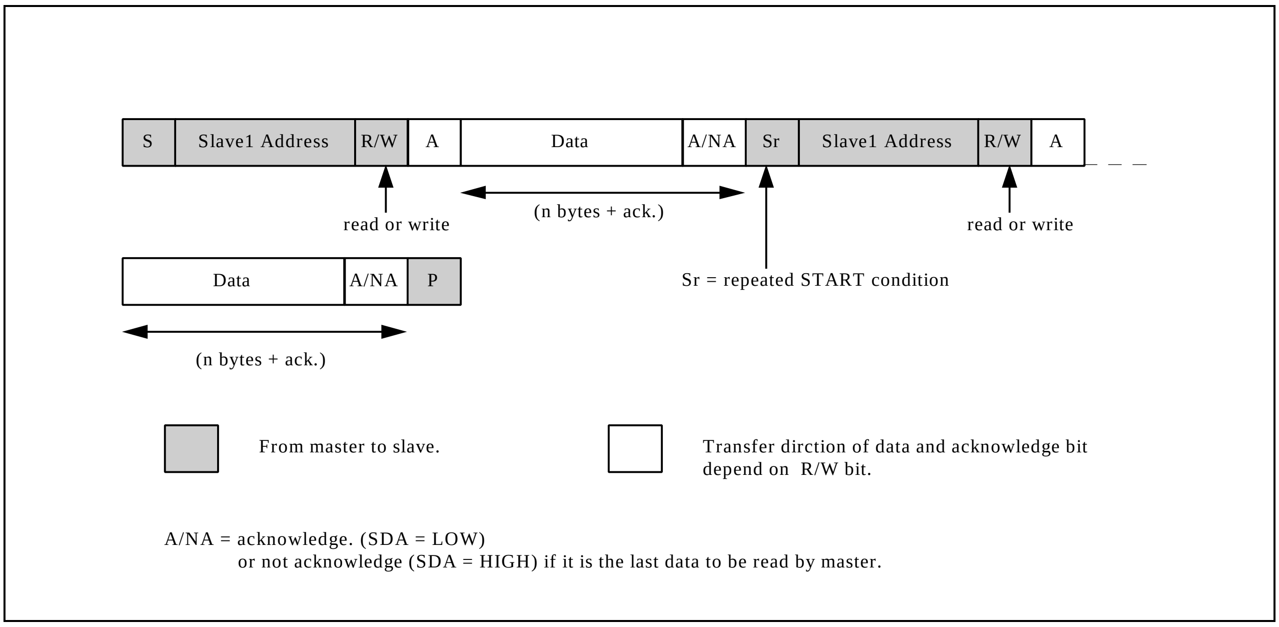
S
Slave1 Address
R/W
A
Data
A/NA
Sr
Slave1 Address
R/W
A
read or write
(n bytes + ack.)
read or write
Data
A/NA
P
Sr = repeated START condition
(n bytes + ack.)
From master to slave.
Transfer dirction of data and acknowledge bit depend on R/W bit.
A/NA = acknowledge. (SDA = LOW)
or not acknowledge (SDA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

Figure 4:

A combined data transfer format for I^2C-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^2C-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^2C-bus to operate properly.

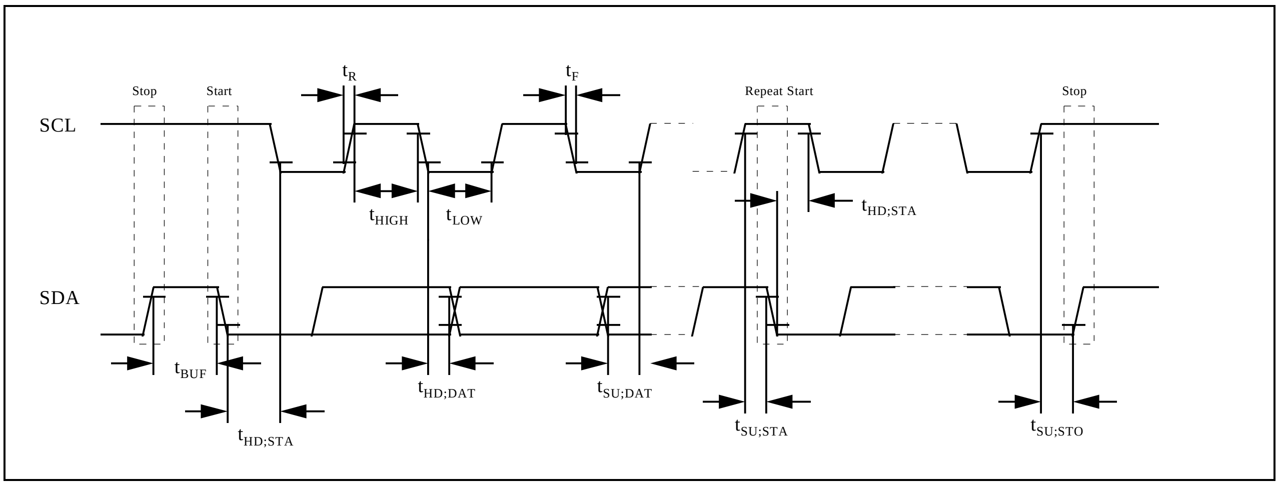

Figure 5:

I^2C-bus timing diagram

Table 1:

Abbreviation for I^2C-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.

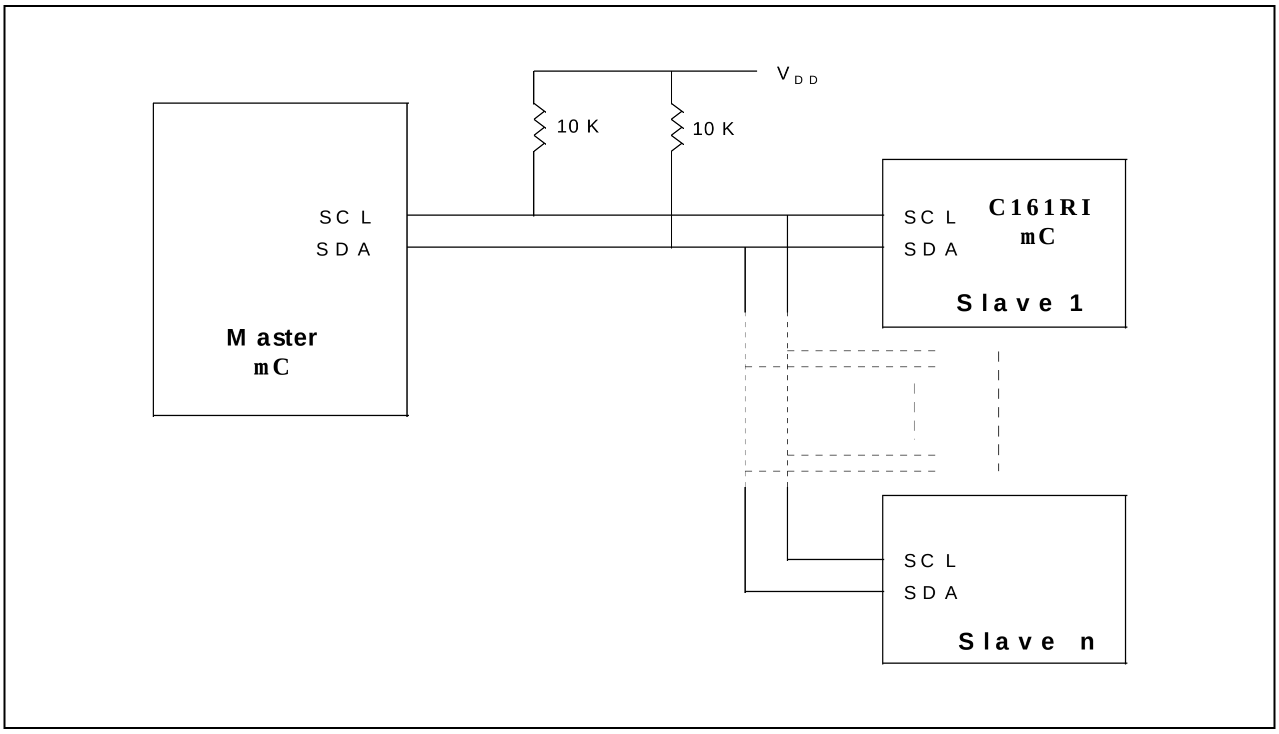

Figure 6:

Hardware connection among master and slave devices

3 Software Description

3.1 Software Concept

The C161RI microcontroller comes with internal hardware I^2C-bus interface. In this software, the hardware I^2C-bus interface of C161RI is configured in the slave mode operation. The clock is generated by the external master controller. The C161RI can support clock frequency of the I^2C-bus up to 400 KHz with 16MHz CPU of the microcontroller.

The I^2C-bus inteface of C161RI consists of two channels with multipexed operation. In the slave mode operation, only one of channel can be selected for operation. There are two types of software design methods which can be used in the slave mode operation. The first one is using the software polling method, and the other is using the hardware interrupt method.

3.1.1 Software polling method

The name of this software module is called SLAVE4.C. The hardware I^2C-bus is needed to be initialized first. There are two channels which has to be selected for operation, and the selected channel of the I^2C -bus must be configured as an open drain output. This software module has been selected in the slave mode operation. Lastly, the slave address has to be assigned for identification.

The interrupt request bit for protocol events will be set if the slave address has been matched. If the read/write bit of the addressing byte is in read operation, the slave will be in slave transmission mode. Then the data byte can be downloaded to the transmit buffer.

The interrupt request bit for data transfer evets will be set if the acknowledge bit of a byte has been received or transmitted. If it is slave transmission, continue download those data. If it is slave reception, store the data to array buffer. The transfer operation will be stopped after the stop condition has been received.

3.1.2 Hardware interrupt method

The name of this software module is called SLAVE2.C. The initialization of the I^2C-bus is similar to the polling method in the above. The additional initialization is to assigned the group and priority level for the software controlled interrupt classes. Finally, enable those two interrupt nodes. One of interrupt vector is the I^2C interrupt request for protocol events. the other interrupt vector is the the I^2C interrupt request for data transfer events.

The interrupt service routine of protocol events will occur when C161RI has received its slave address, or arbitration lost has occurred. The arbitration lost will occur when there is a „high“ on the last received bit (LRB). In this case, the arbitration loss bit (AL) must be cleared via software. Please note that the AL bit cannot be cleared directly. The AL bit can only be cleared by reseting the LRB and IRQP bit simultaneously. If the AL bit is not cleared, the clock and data lines will be pulled down to low permanately after the reception of the slave address.

The interrupt service routine of data transfer evets will occur after the acknowledge bit of a byte has been received or transmitted. If it is slave transmission, continue download those data. If it is slave reception, store the data to array buffer.

3.2 Description of Module Subroutines

3.2.1 Software polling method

I^2C-BUS Software Module

Source file:	`SLAVE4.C`
Header file:	`REG161RI.H`
User definition file:	`I2C161.DEF`

Description

This software module is for the hardware I^2C-bus in the slave mode operation by using polling method for C161RI microcontroller. The clock and as well as transmit/receive data are handled by the internal hardware peripheral of the I^2C-bus.

Module Subroutines

1. `void iic_slave_init();`
2. `void i2cSlaveMode();`

`void iic_slave_init()`

Initialization the hardware peripheral of the I^2C-bus in slave mode. Configurate the I/O port in open drain operation, and assign the slave address.

Parameter	Description
None	

`void i2cSlaveMode()`

Do the software polling for interrupt request bit of protocol events, and interrupt request bit of data transfer events.

Parameter	Description
None	

3.2.2 Hardware interrupt method

I^2C-BUS Software Module

Source file:	`SLAVE2.C`
Header file:	`REG161RI.H`
User definition file:	`I2C161.DEF`

Description

Module Subroutines

1. `void iic_slave_init();`
2. `void data_transfer_interrupt(void) interrupt 0x40;`
3. `void iic_proto_interrupt(void) interrupt 0x41;`

`void iic_slave_init()`

Initialization the hardware peripheral of the I^2C-bus in slave mode. Configurate the I/O port in open drain operation, assign the slave address, assigned the group and priority level for the software controlled interrupt classes, and enable those two interrupt nodes.

Parameter	**Description**
None	

`void data_transfer_interrupt(void) interrupt 0x40`

The interrupt service routine of data transfer evets will occur after the acknowledge bit of a byte has been received or transmitted.

`void iic_proto_interrupt(void) interrupt 0x41`

The interrupt service routine of protocol events will occur when C161RI has received its slave address, or arbitration lost has occurred.

4. Software Listing

Those two software modules are having include files of „reg161ri.h“, and „i2c161.def. The „reg161ri.h“ consists of the special function register (SFR) of hardware I^2C-bus which comes with the compiler. The „i2c161.def“ consists of the location for the SFR bit of hardware I^2C-bus.

Listing of „i2c161.def“

```
#define LRB         0x0008
#define IRQD        0x0020
#define AL          0x0002
#define BB          0x0010
#define SLA         0x0004
#define IRQP        0x0040
#define TRX         0x0080
#define ACKDIS      0x0020
```

4.1 Software polling method (SLAVE4.C)

```
/***************************************************************************/
/*                                                                         */
/*                 SIEMENS Standard Software                               */
/*                                                                         */
/*                 Unauthorized copying prohibited                         */
/*                                                                         */
/*==================================================== */
/*        Programmer:    Tan Choon Hock                                    */
/*        Department:    Siemens SCPL HLM                                  */
/*        Revision :     1.0                                               */
/*                                                                         */
/*==================================================== */
/*                                                                         */
/* This software routine is for the hardware I2C-bus of C161RI             */
/* The C161RI is configured in the slave mode operation.                   */
/* This module is designed to receive data from master as well             */
/* as transmit data to master. This module is using polling                */
/* method to transmit and receive data.                                    */
/*                                                                         */
/***************************************************************************/

#include <reg161ri.h>
#include <intrins.h>
#include "i2c161.def"
```

```
unsigned int buf[10];
unsigned char ptr=0;
unsigned int slave_addr;
const unsigned int txbuf[10] = {0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88,0x99,0x00};


/*****************************************************************************/
/*                                                                           */
/*  Initialization of hardware i2c-bus.                                     */
/*                                                                           */
/*****************************************************************************/
void iic_slave_init()
{
      _bfld_(ODP3,0x0003,0x0003);    // configure P3.0 and P3.1 to open drain output
      _bfld_(DP3,0x0003,0x0003);
       // must configure p3.0 and p3.1 to output for alternative function(i2c)

       ICCFG = 0x0011;                    //enable SDA0 and SCL0
       ICCON = 0x0004;                    //slave mode, 7-bit address
       ICADR = 0x002E;                    //slave address = 2EH

       ptr = 0;

}



/*****************************************************************************/
/*                                                                           */
/* Slave transmission and receiption of data by using software polling     */
/* method.                                                                   */
/*                                                                           */
/*****************************************************************************/
void i2cSlaveMode()
{
      while ((ICST & SLA) && (ICST & BB))
      {
         if (ICST & IRQP)
         {
           slave_addr = ICRTB;           //capture slave address

           ICST &= ~IRQP;                //IRQP = 0
           ICCON &= ~TRX;

           ptr = 0;
```

```
            if (slave_addr != ICADR)
            {
              ICRTB = txbuf[ptr];           //send data if R/W bit = 1
              ptr++;
            }
         }

         if (ICST & IRQD)
         {
             if (slave_addr == ICADR)
                 buf[ptr] = ICRTB;          //capture data if R/W bit = 0

             else
             {                              //send data if R/W bit = 1
               if (ICST & LRB)              //check for last byte to be sent
               {                            //ack = 1
                 ICCON &= ~TRX;             //to release data line
                  ICST &= ~IRQD;
               }
               else ICRTB = txbuf[ptr];  //continue sending data if the ack = 0
             }

            ptr++;

         }
      }
}


void main()
{
       iic_slave_init();

       IEN = 1;

       while (1)
       {
         i2cSlaveMode();
       }
}
```

4.2 Hardware interrupt method (SLAVE2.C)

```
/*****************************************************************************/
/*                                                                           */
/*                SIEMENS Standard Software                                  */
/*                                                                           */
/*                Unauthorized copying prohibited                            */
/*                                                                           */
/*=========================================================================== */
/*          Programmer:    Tan Choon Hock                                    */
/*          Department:     Siemens SCPL HLM                                 */
/*          Revision  :         1.0                                      */
/*                                                                           */
/*=========================================================================== */
/*                                                                           */
/*     The C161RI is configured in the slave mode operation.         */
/*     This module is designed to receive data from master (slave            */
/*     receiver) as well as transmit data to master (slave                   */
/*     transmitter) by uisng hardware interrupt method.                      */
/*                                                                           */
/*****************************************************************************/

#include <reg161ri.h>
#include <intrins.h>
#include "i2c161.def"


unsigned int buf[10];
unsigned char ptr=0;
unsigned int slave_addr;
const unsigned int txbuf[10] = {0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88,0x99,0x00};


/*****************************************************************************/
/*     Subroutine:     iic_slave_init                                        */
/*                                                                           */
/*     Description:    Initialize hardware i2c peripheral, and interrupt     */
/*                     settings.                                             */
/*                                                                           */
/*     Input:           None                                                 */
/*                                                                           */
/*     Return:          None                                                 */
/*                                                                       */
/*****************************************************************************/

void iic_slave_init()
{
        _bfld_(ODP3,0x0003,0x0003);   // configure P3.0 and P3.1 to open drain output
        _bfld_(DP3,0x0003,0x0003);
        // must configure p3.0 and p3.1 to output for alternative function(i2c)

        ICCFG = 0x0011;               //enable SDA0 and SCL0
        ICCON = 0x0004;               //slave mode, 7-bit address
        ICADR = 0x002E;               //slave address = 2EH

        XP0IC = 0x0005;               //ILVL = 1, GLVL = 1
        XP0IE = 1;                    //enable interrupt

        XP1IC = 0x0006;               //ILVL = 1, GLVL = 2
        XP1IE = 1;                    //enable interrupt

        ptr = 0;

}


/*****************************************************************************/
/*      Interrupt Service Routine:     data_transfer_interrupt               */
/*                                                                           */
/*      Description:  The interrupt occurs after the acknowledge bit */
/*                       for a byte has been received or transmitted.        */
/*                                                                   */
/*      Input:   One byte of data to be sent to master device if it is       */
/*               slave transmitter. One byte of data to be received          */
/*               from master if it is slave receiver.                        */
/*                                                                           */
/*      output:    Acknowledge require:                                      */
/*                  Generate LOW output after a byte is received             */
/*                                                                           */
/*****************************************************************************/
void data_transfer_interrupt(void) interrupt 0x40
{

        if (slave_addr == ICADR)
        {
          buf[ptr] = ICRTB;           //capture data if R/W bit = 0
          ICST &= ~IRQD;
        }

        else
        {
           if (ICST & LRB)            //check for last byte to be sent
           {                          //ack = 1, stop sending data
```

```
            ICCON &= ~TRX;              //to release data line
            ICST &= ~IRQD;
        }

        else
        {
            ICRTB = txbuf[ptr];         //continue sending next data if the ack = 0
            ICST &= ~IRQD;
         }

      }

      ptr++;

}


/*******************************************************************************/
/*    Interrupt Service Routine:     iic_proto_interrupt                        */
/*                                                                              */
/*    Description:  The interrupt occurs when the microcontroller               */
/*                    has been selected as a slave device. In other             */
/*                   word, the device address has been received                */
/*                   which matches with the ICADR.  Also when                   */
/*                   there is an arbitration lost.                              */
/*                                                                              */
/*    Input:         The device address has been received.                     */
/*                                                                              */
/*    Output:       Generate an acknowledge bit. If it is slave                 */
/*                   transmitter, start sending the first byte of data.         */
/*                                                                              */
/*******************************************************************************/
void iic_proto_interrupt(void) interrupt 0x41
{

      slave_addr = ICRTB;               //capture slave address

      ptr = 0;                          //reset pointer

            if ((ICST & LRB) && (ICST & AL))
         ICST &= ~(LRB | IRQP);//by reseting LRB and IRQP, it will
                  //automatically reset the AL bit (arbitration lost)
      else
      {
        ICST &= ~IRQP;                  //IRQP = 0
```

```
           if (slave_addr != ICADR)
           {
             ICRTB = txbuf[ptr];               //send data if R/W bit = 1
             ptr++;
            }
        }


}



void main()
{

        iic_slave_init();

        IEN = 1;

        while (1);

}
```