Implementing XSD Host Using a GPIO

Application Note *February 9, 2005* *AN1167.0*

Introduction

The ISL6296 uses the XSD single-wire serial bus to communicate with a host microprocessor. This application note describes how to implement the XSD bus host using a single GPIO pin of the microprocessor. The XSD bus host can also be implemented using a UART. The difference between the two methods is how a bit is received or transmitted. All algorithms at byte or higher level are the same.

Typically there are two registers in the microprocessor related to a GPIO port (of multiple GPIO pins). A data direction register controls the direction (input or output) of each GPIO pin of that port. If a GPIO pin is an output pin, the value of the GPIO pin (either ‘1’ or ‘0’) is latched in a data register. Writing to the register changes the value in the data register and, hence, the GPIO pin output. Reading the data register returns the value of the data register, which is the same value as the GPIO output. If the GPIO pin is an input pin, reading the data register returns the digital value applied to the GPIO pin.

Implementing the XSD host requires only one GPIO pin. Figure 1 shows the circuit diagram. The XSD transmitter is required to be an open-drain output. When it is sending a ‘low’ signal, it pulls the XSD bus to low. When sending a ‘high’ signal, it leaves the XSD bus floating so that the external pull-up resistor pulls the bus voltage to high. Some microprocessors do not have an open-drain output. To deal with such an issue, the GPIO pin can be set as an input pin when transmitting the ‘high’ signal, to avoid actively driving the XSD bus to high. Transmitting the ‘low’ signal is straightforward, just set the GPIO pin as an output and write a ‘0’ to the data register.

The XSD bus transaction consists of transmitting and/or receiving of multiple bytes of data. Each byte is made of 8 bits. The following explains how a bit is transmitted or received, followed by how a transaction is executed. The implementation uses bus voltage polling to transmit or receive a bit; hence, any interrupt function should be disabled, unless the interrupt is very critical.

Sending a Bit

The bit values are determined by the timing of the rising edge. Figure 2 shows the timing diagram. The ‘break’ signal is a special signal and will be discussed later.

Sending a bit is straightforward. Figure 3 shows the flow chart of the operation. For sending a digital ‘1’, the host first drives the GPIO pin to low, delays for 0.3BT, and then releases the GPIO by setting the pin as an input and waits for 0.7BT. For sending a ‘0’, the first delay (delay A) is changed to 0.7BT and the second delay (delay B) is 0.3BT.

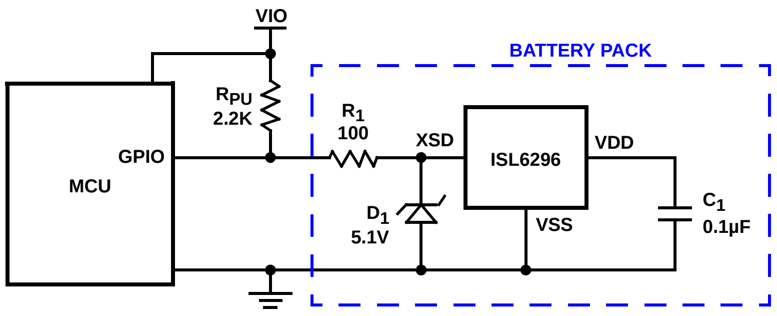

FIGURE 1. THE INTERFACE CIRCUIT USING A SINGLE GPIO PIN

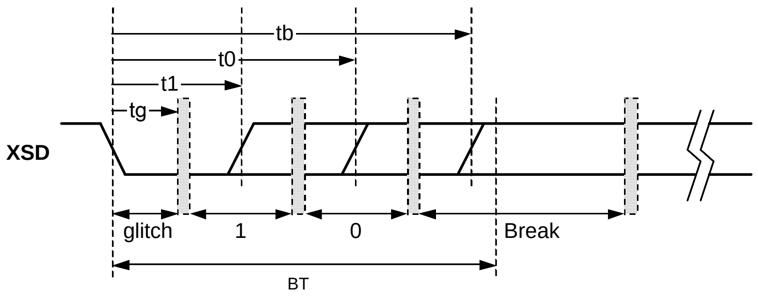

FIGURE 2. THE BUS SIGNAL TIMING DIAGRAM

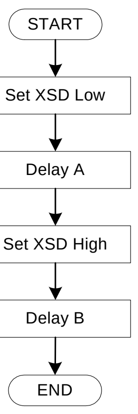

FIGURE 3. FLOW CHART FOR WRITING A BIT TO THE XSD BUS

A sample code written in C language for writing a bit to the XSD bus is given in the Appendix.

The host can add in error checking capability to the XSDWriteBit() subroutine (see the Appendix) while waiting for bit time to finish, after releasing the bus. Figure 4 shows the flow chart with the optional error checking function. A small delay after releasing the bus, the host checks whether or not the bus does rise to high. If it does not, then the subroutine returns a bus error; otherwise, it returns a success code. The C code for writing a digital bit with error checking is also given in the Appendix.

Sending a ‘Break’

There are two types of break signal the host can send. The power-on break is a short break that has a pulse width of between 20μs to 35μs. A regular break has a pulse width of 1 to 100 of the bit time (BT). It is recommended to use 2 to 10 BT for the regular break.

Receiving a Bit

Since the XSD bus transaction is always initiated by the host, the host can poll the XSD bus after sending the instruction frame. Figure 5 shows the flow chart for the receiving function. The host continuously monitors the XSD bus for a falling edge. If the host cannot detect a falling edge within a given TIMEOUT limit, it returns with a BitTIMEOUT error code. Once the falling edge is detected, the host samples the XSD bus value after delay F, which is recommended to be 20μs. If the XSD bus value is ‘high’, the detected falling edge is a glitch. If the bus rises after 0.3BT but before 0.7BT, the received data is digital ‘1’. If the rising edge occurs at 0.7BT, the received bit is ‘0’. If the rising edge happens at 1.4BT, then a ‘break’ is received. If certain time after 1.4BT, the rising edge still does not happen, there is a problem with the bus and a BusErr code is returned.

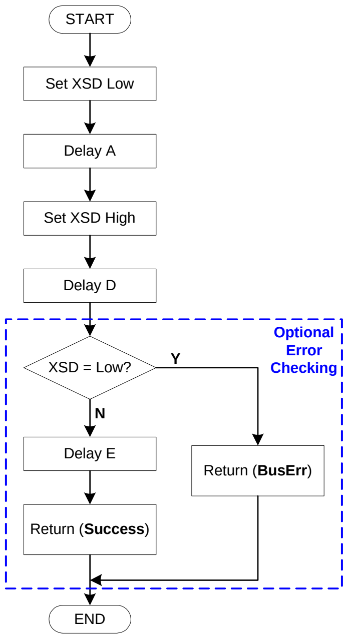

FIGURE 4. FLOW CHART FOR WRITING A BIT TO THE XSD BUS WITH ERROR CHECKING

Receiving a ‘Break’

The ISL6296 and future Intersil single-wire devices may send a ‘break’ signal to the host for emergency indication. In order for the microprocessor to respond to an unexpected

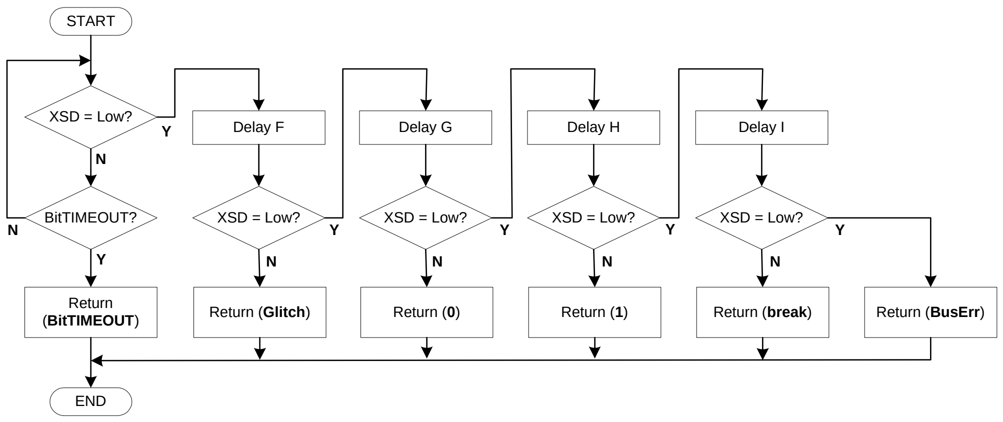

FIGURE 5. THE FLOW CHART FOR READING A BIT FROM THE XSD BUS

'break' signal, a falling-edge signal on the GPIO pin should trigger an interrupt. This interrupt should be disabled during normal polling period of the bus transaction and enabled when the bus is not expected to have transactions.

Sending a Byte

A byte consists of eight bits. The XSD bus sends the LSB (Least Significant Bit) first. The subroutine XSDWriteByte() shows how a byte is transmitted. The 8-bit data is passed to the subroutine through the variable. This example calls the XSDWriteBitwErrChk() subroutine. When an error occurs during the transaction, the XSDWriteByte() subroutine returns an error code. Otherwise, it returns a successful code. The XSDWriteBit() can also be called for simplicity, if error checking capability is not important.

Reading a Byte

The XSDReadByte() function calls the XSDReadBit() eight times to read the byte data. The result byte data is passed through the argument. If any error happens during the transaction, an error code is returned by the function call.

Timing Specifications

The timings used in the subroutines are dependent on the bus speed, or bit time (BT). Table 1 and Table 2 specify the timings.

Table 1 is the timing specification for writing a bit without the error checking capability. To write a '1' to the XSD bus, the host forces the bus to 'low', waits for the delay A for '1', and then releases the bus. The delay A can have a large tolerance, but should be implemented as close to the typical value as possible. The unit for the table is the nominal bit time (BT_N). For example, if the selected bus speed is 5.78kHz, the BT_N = 1/5.78kHz = 173μs. The typical delay A for '1' is 0.3X 173 = 51.9μs. The delay B is the delay from the rising edge to the end of the transmitting bit time. Counting from the starting point of the bit time for the timing specification reduces the cumulative error and therefore is more accurate.

The timing specification for writing a bit with error checking, as shown in Figure 4, can also use the specification in Table 1. The delay A has the same specification as the values in Table 1. The delay D is a small delay. A recommended value is 10 to 20μs. Delay (D + E) has the same value as delay B.

Table 2 is the timing specification for the flow chart shown in Figure 5. The maximum values are affected by the polling (sampling) interval before the XSD bus voltage falls (t_1 - t_0, as shown in Figure 7). If this interval is small compared to the bit time, it can be neglected.

TABLE 1. TIMINGS FOR WRITING A BIT WITHOUT ERROR CHECKING

DELAY	MIN (BT_N)	TYP (BT_N)	MAX (BT_N)	COMMENTS
A for '1'	0.19	0.3	0.43	
A for '0'	0.57	0.7	0.82	
A + B for '1'	0.67	1.0	2.0	
A + B for '0'	1.15	1.15	2.0	

TABLE 2. RECOMMENDED DELAY TIMES FOR READING XSD BUS

DELAY	MIN	TYP	MAX	COMMENTS
F	20μ**s**		25μ**s**	t_1 to t_2
G	0.34BT_N	0.4BT_N	0.62BT_N - (t_1-t0)	t_1 to t_3
H	0.77BT_N	0.8BT_N	0.9BT_N -(t_1-t0)	t_1 to t_4
I	1.53BT_N	1.6BT_N		t_1 to t_5

Instruction Frame

An instruction frame is a two-byte code that contains the information of the device address, OPCODE (operation code), BANK, EEPROM or register address, and the total data byte count of the transaction. Figure 8 shows the bit definition of the instruction frame. Refer to the datasheet for more information. To send the instruction frame to the ISL6296, the host calls the XSDWriteByte() function twice. The first time is to send the lower byte (LoByte) and the second time is to send the higher byte (HiByte).

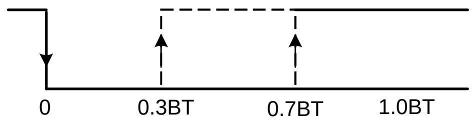

FIGURE 6. TIMING DIAGRAM FOR WRITING A BIT TO THE XSD BUS

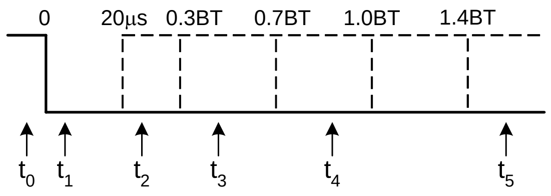

FIGURE 7. TIMING DIAGRAM FOR READING A BIT FROM THE XSD BUS

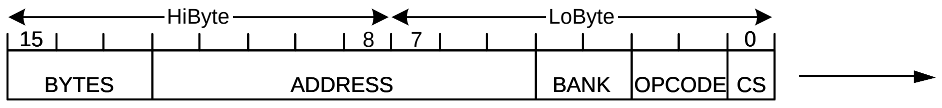

FIGURE 8. INSTRUCTION FRAME

HiByte								LoByte							
15							8	7							0
BYTES			ADDRESS								BANK		OPCODE		CS
0	1	0	0	0	0	0	X	X	X	0	0	0	0	0	X

FIGURE 9. INSTRUCTION FRAME FOR WRITING TO EEPROM

Writing Data to EEPROM

Before the secret address is locked out, the host can write to any even number address, two bytes at a time. The only exception is the location for the default trimming (0-01: DTRM) that is a read-only address. Writing to DTRM will be ignored. The instruction frame for writing to the EEPROM transaction is shown in Figure 9. The CS bit is either ‘0’ or ‘1’, depending on the device address. The default value when the device is shipped from the manufacture is ‘0’. Since only even numbered addresses can be written and only two bytes can be written every time, the LSB of the address field is always ‘0’ and the BYTE field is always ‘010’. To write two-byte data to the EEPROM, the host first sends the two-byte instruction frame, then the byte that goes to the address 00-0000XXX0 and last the byte goes to the address 00-0000XXX1. The XSDWriteByte() function is called four times. An C-code example is given in this application note. This example does not include the bus error handling capability but can be added in with the error handling capability that is already built-in in the XSDWriteByte() function. It is important to point out that writing two bytes of data to the EEPROM takes about 1.8ms. No bus activity should happen during the 1.8ms; therefore, in the example code, a 2ms delay is introduced before the subroutine finishes.

Reading Data from EEPROM

There are two operations to read data from the EEPROM, one with a CRC byte at the end (OPCODE = 11) and the other one does not have the CRC byte (OPCODE = 01). The instruction frame is similar to the one shown in Figure 9, with the OPCODE field replaced with the read codes. Since reading from the EEPROM is allowed to have up to 16 bytes at a time, the BYTES field varies. Reading EEPROM involves two times of XSDWriteByte() function call followed by a number of the XSDReadByte() function call. The number is the same as the total expected number of bytes to be read. A C-code example is given in this application note without the error handling capability for simplicity.

Authentication

The authentication process involves two actions. One is to challenge the device and read back the hash result from the device. The second one is to calculate the expected hash result on the host itself. Both calculations are based on the same 64-bit secret code and the 32-bit challenge code. The calculation on the host side will be covered by a separate application note. This section addresses how to challenge and read the hash result from the ISL6296.

Three steps are involved to read the hash result from the ISL6296:

1. Select the secrets from the three sets of secret stored in the addresses between 0-02 to 0-0D. This step requires a write transaction to address 2-00 (SESL register) with one byte of data.
2. Write the challenge code to the ISL6296. This step requires a write transaction to address 2-01 (CHLG register) with four bytes of data. The four-byte data is the challenge code.
3. Read the result from the AUTH register. This step involves a read transaction from address 2-05 (AUTH register) with one byte of data.

The above three steps are required every time an authentication is performed.

Summary

This application described in detail how to and how easy to implement an XSD bus host using a single GPIO pin. The XSD host is capable of checking errors, such as that caused by the XSD bus being shorted to ground or the XSD device being disconnected from the bus. This application note also provides examples of how to write data to EEPROM, how to read data back from the device, and how the authentication process is executed. Example C-codes are provided in the appendix.

Appendix: Example C-Code

```
// bit 7 of Port A is used as the XSD bus
//PADDR is the data direction register address for port A
//PADR is the data register address for port A
#define  SetXSDAsOutput      PADDR |= 0x80; PADR &= 0x7F
#define  SetXSDAsInput       (PADDR &= 0x7F)
#define  XSD                 (PADR & 0x80)

//The following defines the delay codes for 5.78KHz bus speed
#define delayA       12
#define delayB       31
#define delayC       23
#define delayD       1
#define delayE       5
#define delayF       0
#define delayG       12
#define delayH       20
#define delayI       40
#define delayJ       5
#define delayK       240
#define delayL       240
#define delayM       240
#define TIMEOUT 5000


//The following defines the error codes during bus communication
#define Zero                 0
#define One                  1
#define Success        2
#define BusErr       3
#define BitTIMEOUT      4
#define Glitch       5
#define XSDBreak       6

#define BYTE         unsigned char

extern void Delay(BYTE time);
extern void XSDWriteBit(BYTE bit);
extern BYTE XSDWriteBitwErrChk(BYTE bit);
extern void XSDPoweronBreak(void);
extern void XSDRegularBreak(void);
extern BYTE XSDReadBit(void);
extern BYTE XSDReadBitwTimingIndication(void);
extern BYTE XSDWriteByte(BYTE data);
extern BYTE XSDReadByte(BYTE *data);
extern BYTE XSDWriteEEPROM(BYTE HiByte, BYTE LoByte, BYTE Byte1, BYTE Byte2);
extern BYTE XSDReadEEPROM2(BYTE HiByte, BYTE LoByte, BYTE *Byte1, BYTE *Byte2);
extern BYTE XSDAuthentication(BYTE CS, BYTE SESL, BYTE CHLG1,BYTE CHLG2, BYTE CHLG3, BYTE CHLG4, BYTE *AUTH);
```

```
/*---------------------------------------------------------------------------------------------------------------
ROUTINE NAME : Delay()
INPUT/OUTPUT : time
DESCRIPTION  : This routine introduces some delay. The delay time is not proportional to
                         the variable (time) because of the extra codes that need be excuted for
                         loading the routine.
---------------------------------------------------------------------------------------------------------------*/
void Delay(BYTE time)
{
        BYTE i;
        for (i = 0; i < time; i++);
}



/*----------------------------------------------------------------------------
ROUTINE NAME : XSDWriteBit()
INPUT/OUTPUT : bit
DESCRIPTION  : Send a bit to the XSD bus. Before the calling this routine, the
                         data register value for the corresponding GPIO pin is already
                         set to '0'
----------------------------------------------------------------------------*/
void XSDWriteBit(BYTE bit)
{
        if (bit)
        {       //Write bit '1'
                SetXSDAsOutput;    //since the data register value is alrady '0',
                                   // setting XSD pin as output drives the GPIO
                                   //pin to LOW.
                Delay(delayA);     //delay A determines the rising edge. For
                                   //5.78kHz bus speed.
                                   //delay A should be 127us (0.7BT)
                SetXSDAsInput;     //Settig the GPIO as input resulting a floating
                                   // GPIO pin. The XSD bus voltage is then pulled
                                   // up by the resistor.
                Delay(delayB);    //delay B is to complete the bit time (BT). For
                                   //5.78kHz speed, delay B should be 55us (0.3BT).
        }
        else
        {       //Write bit '0'
                SetXSDAsOutput;
                Delay(delayB);    //delay 0.3BT
                SetXSDAsInput;
                Delay(delayA);    //delay 0.7BT
        }
}
```

```
/*------------------------------------------------------------------------------
ROUTINE NAME : XSDWriteBitwErrChk()
INPUT/OUTPUT : bit
DESCRIPTION  : Send a bit to the XSD bus with error checking capability.
               Before the calling this routine, the data register value for
               the corresponding GPIO pin is already set to '0'.
               The return value indicates either the transitting is successful
               or failed.
------------------------------------------------------------------------------*/
BYTE XSDWriteBitwErrChk(BYTE bit)
{
        if (bit)
        {       //Write bit '1'
                SetXSDAsOutput;    //since the data register value is alrady '0',
                                   // setting XSD pin as output drives the GPIO
                                   //pin to LOW.
                Delay(delayA);     //delay A determines the rising edge. For
                                   //5.78kHz bus speed.
                                   //delay A should be 127us (0.7BT)
                SetXSDAsInput;     //Settig the GPIO as input resulting a floating
                                   // GPIO pin. The XSD bus voltage is then pulled
                                   // up by the resistor.
                Delay(delayD);
                if (XSD == 0) return (BusErr);    //XSDBusErr needs be defined.
                Delay(delayC);                     //XSD defined as (PADR & 0x80),
        }                          //which is the bit 7 of PA
        else
        {       //Write bit '0'
                SetXSDAsOutput;
                Delay(delayB);     //delay 0.3BT
                SetXSDAsInput;
                Delay(delayD);     //delay 0.7BT
                if (XSD == 0) return (BusErr);
                Delay(delayE);
        }
        return (Success);      //Success needs be defined.
}


/*--------------------------------------------------------------------------------------------------------
ROUTINE NAME : XSDPoweronBreak()
INPUT/OUTPUT : None
DESCRIPTION  : Send a 30us break signal
--------------------------------------------------------------------------------------------------------*/
void XSDPoweronBreak(void)
{
        SetXSDAsOutput;    //since the data register value is alrady '0',
                           // setting XSD pin as output drives the GPIO
                           //pin to LOW.
        Delay(delayJ);     //delay 30us
        SetXSDAsInput;     //Settig the GPIO as input resulting a floating
        Delay(delayK);    //Additional delay, could be as short as 0us.
}

/*--------------------------------------------------------------------------------------------------------
ROUTINE NAME : XSDRegularBreak()
INPUT/OUTPUT : None
DESCRIPTION  : Send a regular break signal.
--------------------------------------------------------------------------------------------------------*/
void XSDRegularBreak(void)
{
        SetXSDAsOutput;    //since the data register value is alrady '0',
                           // setting XSD pin as output drives the GPIO
                //pin to LOW.
        Delay(delayL);     //delay 2BT
        SetXSDAsInput;     //Settig the GPIO as input resulting a floating
        Delay(delayM);    //Additional delay, could be as short as 0us.
}
```

```
/*-----------------------------------------------------------------------------
ROUTINE NAME : XSDReadBit()
INPUT/OUTPUT : bit
DESCRIPTION  : Read the bit value to the variable (bit). The returned value from
               the funtion contains either the XSD bit value or an error code.
-----------------------------------------------------------------------------*/
BYTE XSDReadBit(void)
{
        int TimeCount;          //Count for TIMEOUT
                                //TIMEOUT needs be defined.
        TimeCount = 0;
        while ((TimeCount <= TIMEOUT) && (XSD !=0)) TimeCount++;
        if (TimeCount >= TIMEOUT)
                return (BitTIMEOUT);            //define BitTIMEOUT
        else
        {
                Delay(delayF);      //delay F is 20us
                if (XSD != 0)
                        return (Glitch);        //define Glitch
                else
                {
                        Delay(delayG);
                        if (XSD != 0)  {
                                return (One);  } //define Zero
                        else
                        {
                                Delay(delayH);
                                if (XSD != 0)    {
                                        return (Zero); }//define One
                                else
                                {
                                        Delay(delayI);
                                        if (XSD != 0)       {
                                                return (XSDBreak);}//define XSDBreak
                                        else
                                                return (BusErr);  //define BusErr
                                }
                        }

                }
        }
}

/*-----------------------------------------------------------------------------
ROUTINE NAME : XSDWriteByte()
INPUT/OUTPUT : data
DESCRIPTION  : Write a Byte data to XSD bus, LSB first. The returned value
               contains the error code if an error occurred.
-----------------------------------------------------------------------------*/
BYTE XSDWriteByte(BYTE data)
{
        BYTE i;

    for (i = 0; i < 8; i++)
    {
        if (XSDWriteBitwErrChk(data & 0x01) != Success)
                return (BusErr);
        else
                data >>=1;
    }
    return (Success);
}
```

```
/*------------------------------------------------------------------------------
ROUTINE NAME : XSDReadByte()
INPUT/OUTPUT : data
DESCRIPTION  : Read a Byte data from the XSD bus, LSB first.
               The result is passed through the variable (data).
               The function returns an error code if an error happens.
------------------------------------------------------------------------------*/
BYTE XSDReadByte(BYTE *data)
{
        BYTE i;
        BYTE result;
        BYTE Temp;

        Temp = 0;

    for (i = 0; i < 8; i++)
    {
        Temp >>= 1;
        result = XSDReadBit();
        if (result == Zero)
        {
        }
        else if (result == One)
        {
                Temp |= 0x80;
        }
        else
        {
                return (result);
        }
    }
    *data = Temp;
    return (Success);
}


/*----------------------------------------------------------------------------------
ROUTINE NAME : XSDWriteEEPROM()
INPUT/OUTPUT : HiByte, LoByte, Byte1, Byte2
DESCRIPTION  : Write two bytes of data (Byte1 and Byte2) to the EEPROM.
               The instruction frame is stored in the HiByte and LoByte.
               Bus error can be returned but is not implemented in this example.
               A regular break is send before the instruction frame.
----------------------------------------------------------------------------------*/

BYTE XSDWriteEEPROM(BYTE HiByte, BYTE LoByte, BYTE Byte1, BYTE Byte2)
{
        XSDRegularBreak();      //Send a regular break before the transaction
        XSDWriteByte(LoByte);   //lower 8 bits of the instruction frame
        XSDWriteByte(HiByte);   //higher 8 bits of the instruction frame
        XSDWriteByte(Byte1);    // data byte goes to the even address
        XSDWriteByte(Byte2);            // data byte goes to the odd address
        Delay(0xFF);            // Introduce more than 2ms delay for Write EEPROM
        Delay(0xFF);            // to finish.
        Delay(0xFF);
}
```

```
/*-----------------------------------------------------------------------------------
ROUTINE NAME : XSDReadEEPROM2()
INPUT/OUTPUT : HiByte, LoByte, Byte1, Byte2
DESCRIPTION  : Read two bytes of data (Byte1 and Byte2) from the EEPROM.
               The instruction frame is stored in the HiByte and LoByte.
               Bus error can be returned but is not implemented in this example.
-----------------------------------------------------------------------------------*/
BYTE XSDReadEEPROM2(BYTE HiByte, BYTE LoByte, BYTE *Byte1, BYTE *Byte2)
{
        XSDRegularBreak();      //Send a regular break before the transaction
        XSDWriteByte(LoByte);   //lower 8 bits of the instruction frame
        XSDWriteByte(HiByte);   //higher 8 bits of the instruction frame
        XSDReadByte(Byte1);     // data byte goes to the even address
        XSDReadByte(Byte2);          // data byte goes to the odd address
}

/*-----------------------------------------------------------------------------------
ROUTINE NAME : XSDAuthentication()
INPUT/OUTPUT : CS, SESL, CHLG1, CHLG2, CHLG3, CHLG4, AUTH
DESCRIPTION  : The entire authentication process. The SESL is the secret selection code.
               The CHLG1 to CHLG4 are the challenge codes from LSB to MSB respectively.
               The AUTH is the hash result. CS is the chip selection (or ISL6296 address)
               Bus error can be returned but is not implemented in this example.
-----------------------------------------------------------------------------------*/

BYTE XSDAuthentication(BYTE CS, BYTE SESL, BYTE CHLG1,BYTE CHLG2, BYTE CHLG3, \
                                                    BYTE CHLG4, BYTE *AUTH)
{
        BYTE LoByte;
        BYTE HiByte;

        //Write to the secret select register
        HiByte = 0x20;               //the Higher byte of the instruction frame
        LoByte = 0x10;        //the lower byte of the instruction frame
        LoByte |= CS;      //if the device address is '1', set bit 0 (LSM)

        XSDRegularBreak();      //Send a regular break before the transaction
        XSDWriteByte(LoByte);   //lower 8 bits of the instruction frame
        XSDWriteByte(HiByte);   //higher 8 bits of the instruction frame
        XSDWriteByte(SESL);          // write the secret selection

        //Write to the challenge register (four bytes)
        HiByte = 0x80;        // four bytes to write for challenge code
        LoByte |= 0x20;       //set the address to 0x01. The rest does not change.
        XSDWriteByte(LoByte);   //lower 8 bits of the instruction frame
        XSDWriteByte(HiByte);   //higher 8 bits of the instruction frame
    XSDWriteByte(CHLG1);    //Write the four bytes challenge code
    XSDWriteByte(CHLG2);
    XSDWriteByte(CHLG3);
    XSDWriteByte(CHLG4);

        //Read the authentication result
        HiByte = 0x20;        //one byte to read, address 2-05
        LoByte |= 0x82;       //change the instruction to read transaction
        XSDWriteByte(LoByte);   //lower 8 bits of the instruction frame
        XSDWriteByte(HiByte);   //higher 8 bits of the instruction frame
        XSDReadByte(AUTH);    // data byte goes to the even address
}
```