AN1167
Rev 0.00
February 9, 2005

Implementing XSD Host Using a GPIO

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.

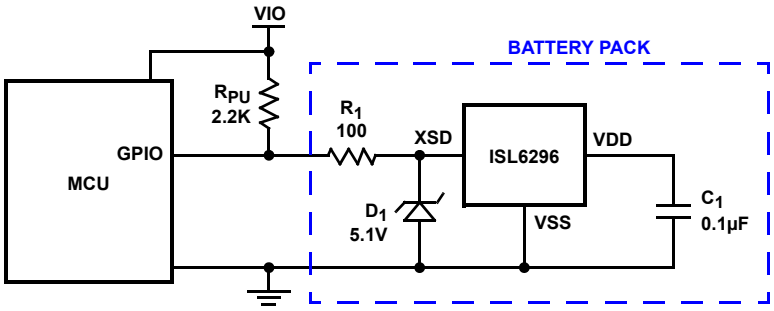

FIGURE 1. THE INTERFACE CIRCUIT USING A SINGLE GPIO PIN

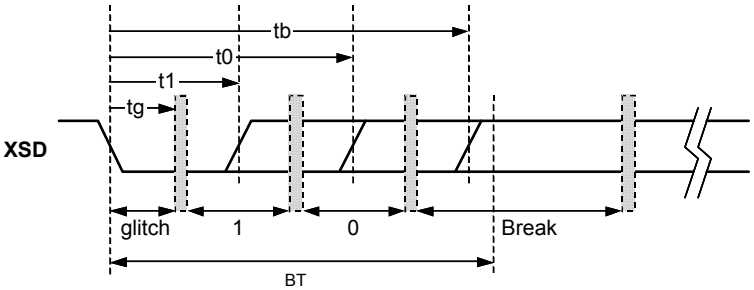

FIGURE 2. THE BUS SIGNAL TIMING DIAGRAM

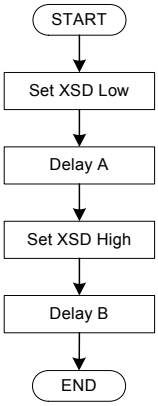

FIGURE 3. FLOW CHART FOR WRITING A BIT TO THE XSD BUS

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

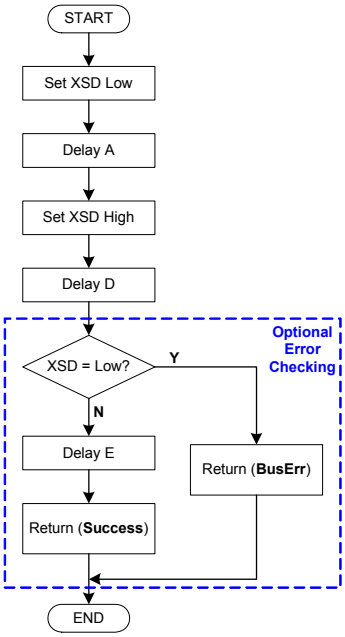

FIGURE 4. FLOW CHART FOR WRITING A BIT TO THE XSD BUS WITH ERROR CHECKING

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.

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

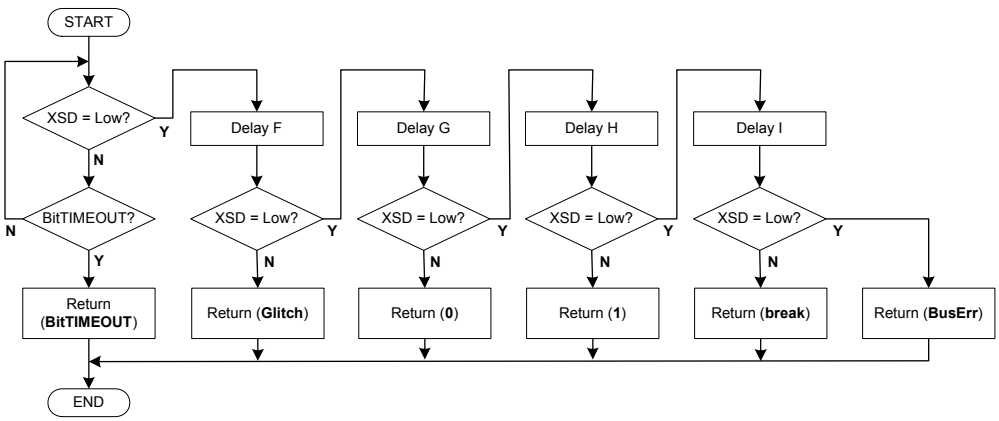

FIGURE 5. THE FLOW CHART FOR READING A BIT FROM THE XSD BUS

'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.

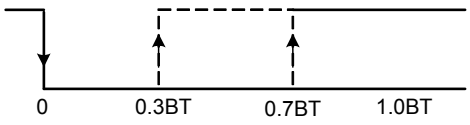

FIGURE 6. TIMING DIAGRAM FOR WRITING A BIT TO THE XSD BUS

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	20µs		25µs	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).

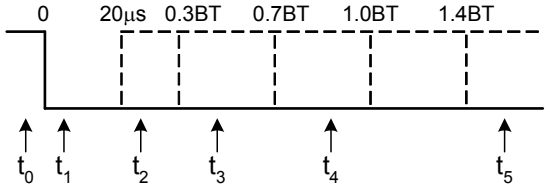

FIGURE 7. TIMING DIAGRAM FOR READING A BIT FROM THE XSD BUS

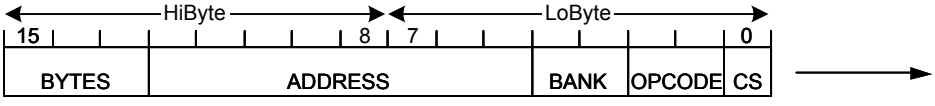

FIGURE 8. INSTRUCTION FRAME

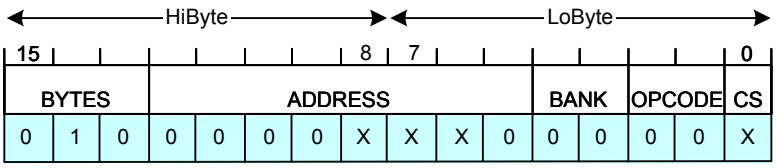

FIGURE 9. INSTRUCTION FRAME FOR WRITING TO EEPROM

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
}
```

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

RENESAS

SALES OFFICES

Renesas Electronics Corporation

http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 7.0