AN10462

SPI programming for Philips Bridge ICs

Rev. 01 — 1 June 2006 Application note

Document information

Info	Content
Keywords	P89LPC935 to SC16IS750, SPI to UART, SPI programming, microcontroller, embedded processor
Abstract	The programming of Philips Bridge ICs such as SC16IS750 through SPI-bus is discussed in this application note. In addition, the source code in C language, containing SPI communication routines between Philips P89LPC935 microcontroller and the SC16IS750 Bridge IC is provided and also discussed. This application note is also applicable to other Bridge ICs such as SC16IS740, SC16IS760, SC16IS752 and SC16IS762.

Revision history

Rev	Date	Description
01	20060601	application note; initial version

Contact information

For additional information, please visit: **http://www.semiconductors.philips.com**

For sales office addresses, please send an email to: **sales.addresses@www.semiconductors.philips.com**

1. Introduction

The Philips Bridge ICs are a new generation of interface solutions for managing high-speed serial data communication among various bus interfaces such as SPI, I^2C-bus, and UARTs including RS-232 and RS-485. The Bridge IC is commonly used to overcome the limitation of the host bus interface to peripherals and provides an easy method to interface with existing different serial bus interfaces.

The description of the hardware connection and firmware programming are described in the next paragraphs for users to quickly understand the implementation of Philips P89LPC935 microcontroller with SPI-bus interface to Philips Bridge IC serial interface for RS-232 point-point communication, RS-485 multi-drop application, IrDA wireless links, and GPIO interface. The source code in C language is provided to show how to write a simple communication program between the microcontroller and the Bridge IC serial interface. The goal is to help users to design the Bridge IC in their application and also shorten their product development cycle.

2. Hardware connection

The block diagram depicted in Figure 1 shows the circuit connection of a Philips Bridge IC such as SC16IS750 interfacing with Philips P89LPC935 microcontroller through the SPI-bus. The Bridge IC offers simple, flexible, and minimal connection to the microcontroller. The Bridge IC allows the microcontroller to easily control with few wires connection through SPI-bus, which is a widely-used serial bus interface.

When the I2C/$\overline{\text{SPI}}$ pin is at logic 0, the SPI-bus interface is selected and the Bridge IC can interface to the microcontroller with a 4-wire connection. The signals on the 4 wires are MISO (Master Input Slave Output), MOSI (Master Output Slave Input), $\overline{\text{CS}}$ (active LOW Chip Select), and SPICLK (SPI Clock).

The SPI-bus is a 4-wire full-duplex synchronous serial data link. Devices connected to the SPI-bus are classified as master or slave devices. The master device initiates the data transfer on the SPI-bus and also controls the slave devices with the chip select pins. The slave devices are active only when selected. The interconnected devices share the same V_{CC} and GND.

After the microcontroller and Bridge IC are wired properly and their connection is established, the microcontroller can send data to, and receive data from, UART devices via the Bridge IC. The Bridge IC receives the SPI data from the microcontroller then sends the data to the following devices: serial devices via RS-232 or RS-485 such as a modem, wireless devices via IrDA such as a remote control, and general-purpose input/output devices such as an LED and keypad. When the Bridge IC receives the data from the devices, it will notify the microcontroller by generating an interrupt ($\overline{\text{IRQ}}$) output signal, then the microcontroller can access the data from the Bridge IC via the SPI-bus interface.

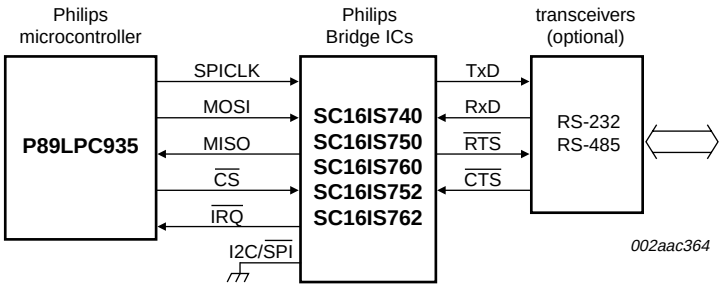

Fig 1. SPI interface to Philips Bridge IC block diagram

3. Firmware programming

The firmware code for the P89LPC935 microcontroller is written in C language. It can be compiled by using an embedded C compiler. The firmware code consists of three major blocks, which are Main Loop program, Interrupt service routine, and Bus interface layer, as described in Section 3.1, Section 3.2 and Section 3.3.

3.1 Main Loop

The Main Loop function is to program the microcontroller's SPI port as a master by initializing the SPI control register, interrupt handler, and chip select as shown in the following sample code:

```
void mcu_spi_init (void) {
     // Set port 2.0 for MOSI, MISO, /SS and SPICLK
     P2M1 &= 0x42;
     P2M2 &= 0x42;

     SPCTL = 0x90;      // Set LPC935 SPI port to Master
     SPCTL &= ~0x08;    // CPOL=0; SCK is low when idle - SPI mode 0
     SPCTL &= ~0x04;    // CPHA=0; shift triggered on leading edge of clock
     SPCTL |= 0x11;     // 00=clk/4, 01=clk/16, 10=clk/64, 11=clk/128

     SPI_SS  = 0;   // Enable slave chip select for SC16IS7x0
     SPI_SS  = 1;   // Disable slave chip select for SC16IS7x0

     // Set SPI interrupt priority to 0
     IP1  &= ~0x08;
     IP1H &= ~0x08;

     ESPI = 1;          // SPI Interrupt
     SPCTL |= 0xC0;     // Enable SPI, ignore SS
}
```

The next step is to initialize the Bridge IC by writing its internal control registers to program, for example, UART baud rate, data line, and FIFO for transmitter and receiver.

```
void init_SC16IS750 (void)   {
     SPI_write (LCR, 0x80);    // 0x80 to program baud rate
     SPI_write (DLL, 0x30);    // 0x30=19.2K, 0x08 =115.2K with X1=14.7456 MHz
     SPI_write (DLM, 0x00);    // divisor = 0x0008 for 115200 bps
     SPI_write (LCR, 0xBF);    // access EFR register
     SPI_write (EFR, 0X10);    // enable enhanced registers
     SPI_write (LCR, 0x03);    // 8 data bit, 1 stop bit, no parity
     SPI_write (FCR, 0x06);    // reset TXFIFO, reset RXFIFO, non FIFO mode
     SPI_write (FCR, 0x01);    // enable FIFO mode
}
```

Inside the Main Loop, the microcontroller can use one of the two methods for communicating to the Bridge IC. The first method is polling the status register of the Bridge IC regularly. It keeps checking the event flags such as data available in receiver, and passes to the appropriate subroutine for further processing.

```
void spi_polling (void)   {
    char data, polling=1;
    while (polling)   {
        if (SPI_read (LSR) && 0x01))  {    // data in receiver
        data = SPI_read (RHR);
        if (data == 27) polling = 0;   // receive ESC char to exit from polling mode
    } }
}
```

The second method is using an interrupt handler in the Interrupt Service Routine until the Bridge IC generates an interrupt output ($\overline{IRQ}$) signal. If using the interrupt handler, the microcontroller and the bus interrupt bits must be enabled and the $\overline{IRQ}$ pin of the Bridge IC must be connected to an external interrupt pin of the microcontroller (for example, the $\overline{INT1}$ pin of the P89LPC935).

3.2 Interrupt Service Routine (ISR)

The microcontroller uses the Interrupt Service Routine to handle an interrupt generated by the Bridge IC. As soon as the Bridge IC generates an interrupt output ($\overline{IRQ}$) signal, the ISR checks the interrupt status of the Bridge IC to determine the interrupt type and sets up proper event flags to inform the Main Loop routine for processing the interrupt request.

```
void mcu_interrupt1(void) interrupt 2 {    // external interrupt 1
     char data;
     switch (SPI_read (IIR) & 0x0f) {      // check interrupt status register
          case 0x06:  // receiver line status (LSR)
                      if (SPI_read(LSR) && 0x01)    // data in receiver
                          data = SPI_read(RHR);
                      break;
          case 0x0C:  // receiver time-out interrupt
          case 0x04:  // receive hold register (RHR) interrupt
                      data = SPI_read (RHR);
                      break;
```

```
            case 0x02:  // transmit hold register (THR) interrupt
                        SPI_write(THR, data);
            default:    break;
        }
}
```

3.3 Bus interface layer

The Bus Interface layer handles the SPI-bus interface between the microcontroller and the Bridge IC for transmitting and receiving the SPI data. The three functions in the bus interface layer are:

SPI_send — The microcontroller transmits data to the SPI-bus and waits until the end of SPI data transmission.

SPI_read — The Bridge IC receives the data from peripherals, then the microcontroller retrieves the data from the Bridge IC vial SPI-bus and stores the data for further processing.

SPI_write — The microcontroller writes data to the Bridge IC via SPI-bus, then the Bridge IC will transmit the data to peripherals.

```
char SPI_send (char byte) {       // mcu sends a byte to spi bus
     SPDAT = byte;                // data is sent
     while(!SPI_tx_completed);    // wait end of transmission
     SPSTAT &= ~0x80;             // clear mcu spi interrupt flag (SPIF)
     SPI_tx_completed = 0;        // clear transmit spi interrupt flag
     return SPDAT;                // receive data on spi read
}

char SPI_read (char register) {   // mcu reads a register from SC16IS750
     SPI_SS = 0;                       // enable slave chip select
     SPI_send((register<<3) | 0x80);   // register address is sent
     register = SPI_send(0);           // dummy data is sent for spi read
     SPI_SS = 1;                       // disable slave chip select
     return register;                  // receive the read data
}

void SPI_write (char address, char data) {  // mcu writes data to SC16IS750
     SPI_SS = 0;                 // enable slave chip select
     SPI_send (address<<3);      // address is sent
     SPI_send (data);            // data is sent
     SPI_SS = 1;                 // disable slave chip select
}
```

4. Conclusion

Philips Bridge IC provides easy interface to a host controller such as Philips P89LPC935 microcontroller and enables seamless high-speed SPI to RS-232 or RS-485 protocols convergence including GPIO for general-purpose input/output and IrDA for wireless links. Also, the Bridge IC offers low voltage operation, low power consumption, and compact design, which is suitable for battery-operated applications. In addition, the Bridge IC reduces software overhead, frees up the host controller's resources, increases design flexibility, and improves overall system performance. For more details about Philips Bridge ICs, please visit the Philips Semiconductors web site to download the data sheets.

5. Abbreviations

Table 1. Abbreviations

Acronym	Description
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter
IrDA	Infrared Data Association
IC	Integrated Circuit
GPIO	General Purpose Input/Output
LED	Light Emitting Diode
FIFO	First In, First Out

6. Legal information

6.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. Philips Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

6.2 Disclaimers

General — Information in this document is believed to be accurate and reliable. However, Philips Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

Right to make changes — Philips Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — Philips Semiconductors products are not designed, authorized or warranted to be suitable for use in medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of a Philips Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. Philips Semiconductors accepts no liability for inclusion and/or use of Philips Semiconductors products in such equipment or applications and therefore such inclusion and/or use is for the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. Philips Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

6.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

I^2C-bus — logo is a trademark of Koninklijke Philips Electronics N.V.

7. Contents

For more information, please visit: http://www.semiconductors.philips.com.
For sales office addresses, email to: sales.addresses@www.semiconductors.philips.com.

Date of release: 1 June 2006
Document identifier: AN10462_1