Semiconductors

AN10041

ISP1109 Firmware Programming Guide

Rev. 01 — 17 May 2005 **Application note**

Document information

Info	Content
Keywords	isp1109, usb, universal serial bus, transceiver
Abstract	This document will help you to utilize the ISP1109 more efficiently.

PHILIPS

Revision history

Rev	Date	Description
01	20050517	First release.

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 ISP1109 is a USB peripheral-only transceiver that supports *CEA-936-A, Mini-USB Analog Carkit Interface*. The operation mode, and status and control registers of the ISP1109 can be configured through the SPI bus or I^2C-bus interface. This firmware-programming guide will help you to utilize the ISP1109 more efficiently.

To a microcontroller or a microprocessor, the ISP1109 appears as a memory device with the SPI bus or I^2C-bus interface.

The organization of this document is as follows:

- Section 2 describes the architecture of the firmware. It also introduces how to port the firmware to other CPU platforms.
- Section 3 describes the hardware abstraction layer.
- Section 4 covers the ISP1109 functional command and implementations of the ISP1109 mode set.
- Section 5 explains the ISP1109 test command and implementations of the ISP1109 test.
- Section 6 focuses on the Interrupt Service Routine (ISR).
- Section 7 describes the Main Loop that execute in the foreground.

2. Architecture

2.1 Firmware structure

The firmware for the evaluation (eval) board consists of five building blocks; see Fig 1.

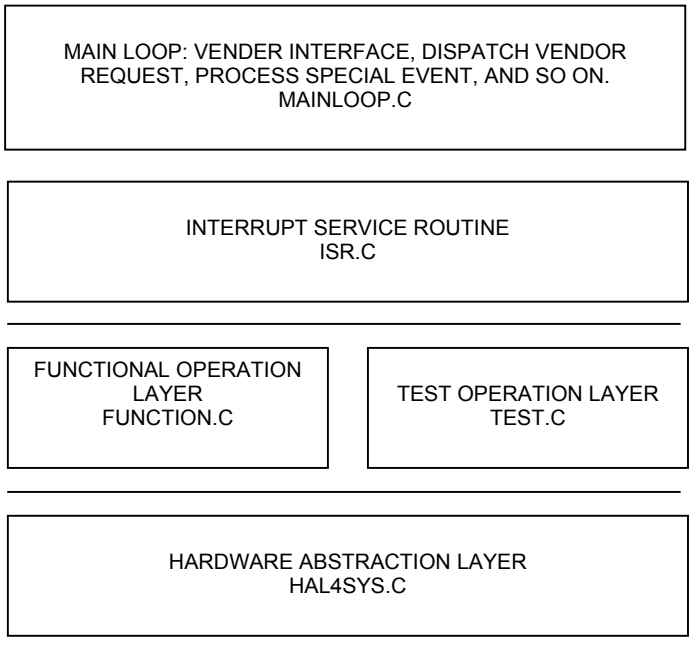

Fig 1. Firmware structure of the ISP1109.

2.1.1 Hardware Abstraction Layer—HAL4SYS.C

This is the lowest layer code in the firmware. It performs hardware dependent I/O access to the ISP1109, as well as the eval board hardware. When porting the firmware to other CPU platforms, this part of the code always needs modifications or additions.

2.1.2 Operation Layer—FUNCTION.C

To simplify functional programming with the ISP1109, the firmware defines a set of command interfaces that encapsulate all the functions used to access the ISP1109.

2.1.3 Operation Layer—TEST.C

To simplify test programming with the ISP1109, the firmware defines a set of command interfaces that encapsulate all the test cases used to access the ISP1109.

2.1.4 Interrupt Service Routine—ISR.C

This part of the code handles interrupts generated by the ISP1109.

2.1.5 Main Loop—MAINLOOP.C

The Main Loop contains the code for human interface, such as key scan. It also checks event flags and passes to the appropriate subroutine for further processing.

2.2 Porting the firmware to other CPU platforms

Table 1: shows the modifications that are required on building blocks.

Table 1: Modifications on building blocks

File name	Test only	Product level
HAL4SYS.C	Port to hardware specific	Port to hardware specific
FUNCTION.C	No change	No change
TEST.C	Depends on test case	No change
ISR.C	No change	No change
MAINLOOP.C	Depends on the CPU and system; ports and interrupt initialization need to be rewritten	Add product specific Main Loop processing

2.3 Using the firmware in polling mode

To use the firmware in polling mode, add the following code in the Main Loop:

```
if(interrupt_pin_low)
fn_usb_isr();
```

Typically, the ISR is initiated by hardware. In polling mode, the Main Loop detects the status of the interrupt pin, and invokes ISR, if necessary.

3. Hardware Abstraction Layer for a system

This layer contains the lowest layer functions that need to be changed, depending on the CPU platform.

```
void Hal4Sys_WaitinUS(IN OUT ULONG time);
void Hal4Sys_WaitinMS( IN OUT ULONG time);
void Hal4Sys_ControlD13Interrupt( BOOLEAN InterruptEN);
UCHAR Hal4Sys_Read(UCHAR address)_
void Hal4Sys_Write(UCHAR data, UCHAR address)_
```

For example:

```
void Hal4Sys_AcquireTimer0(void)
{
Hal4Sys_OldIsr4Timer = getvect(0x8);
setvect(0x8, Hal4Sys_Isr4Timer);
}
void Hal4Sys_End_Int(void) // end of interrupt routine
{
outportb(0x20,0x20);
}
void Hal4Sys_Initial_uP(void)
{
Hal4Sys_SysCtrl_PORT = 0;
Hal4Sys_AcquireTimer0();
}
void Hal4Sys_ResetPCBA(void)
{
Hal4Sys_SysCtrl_PORT |= SCP_RST;
outportb(D13_SYSCTRL_PORT,Hal4Sys_SysCtrl_PORT); //
Hal4Sys_WaitinUS(50);
Hal4Sys_SysCtrl_PORT &= ~SCP_RST;
outportb(D13_SYSCTRL_PORT,Hal4Sys_SysCtrl_PORT); // reset all PCBA
Hal4Sys_WaitinUS(50);
Hal4Sys_SysCtrl_PORT |= SCP_RST;
outportb(D13_SYSCTRL_PORT,Hal4Sys_SysCtrl_PORT); // reset release
}
```

4. Operation Layer for functions of the ISP1109

The following functions are defined as the ISP1109 command interface to simplify programming. These are the implementations of the ISP1109 mode set, which is defined in the ISP1109 data sheet.

```
void Hal4D09_SpeedSuspend_Pin(void);
void Hal4D09_SpeedSuspend_Reg(void);
void Hal4D09_USB_LowSpeed(void);
void Hal4D09_USB_FullSpeed(void);
void Hal4D09_USB_Suspend(void);
void Hal4D09_USB_Active(void);
void Hal4D09_USB_VPVM_U(void);
void Hal4D09_USB_VPVM_B(void);
void Hal4D09_USB_DAT_SE0_U(void);
void Hal4D09_USB_DAT_SE0_B(void);
void Hal4D09_UART1(void);
void Hal4D09_UART2(void);
void Hal4D09_Stereo(void);
void Hal4D09_Mono(void);
```

For example, the implementation of the Set UART Mode command is as follows:

```
void Hal4D09_UART1(void)
{
    Hal4Sys_Write(0x80, 0x05);
```

```
    Hal4Sys_Write(0x40, 0x04);
}
```

5. Operation Layer to test the ISP1109

These functions are defined to verify the ISP1109 silicon. It can be removed for product programs.

6. Interrupt Service Routine

At the entrance of the Interrupt Service Routine (ISR), the firmware uses the Read Interrupt register to decide the source of an interrupt and then dispatches the interrupt to the appropriate subroutines for processing.

```
void interrupt usb_isr(void)
{
   Hal4Sys_ControlD13Interrupt(FALSE);
   fn_usb_isr();
                 Hal4Sys_End_Int();
}
void fn_usb_isr(void)
{
    UCHAR Int_flags;
                  Int_flags = Hal4Sys_Read(0x0a);
                  Hal4Sys_Write(Int_flags, CLR_INT_LATCH);
    if(Int_flags && VBUS_DET)
       Isr_vbus_det_int();
    if(Int_flags && SESS_VLD)
       Isr_sess_vld();
    if(Int_flags && DP_HI)
       Isr_dp_hi();
    if(Int_flags && ID_GND)
       Isr_id_gnd();
    if(Int_flags && SE1)
       Isr_se1_iel();
    if(Int_flags && ID_FLOAT)
       Isr_id_float();
    if(Int_flags && DP_INT)
       Isr_dp_int();
}
void Isr_vbus_det_int(void)
{
}
void Isr_sess_vld(void)
{
}
void Isr_dp_hi(void)
{
}
void Isr_id_gnd(void)
{
}
```

```
void Isr_se1_iel(void)
{
}
void Isr_id_float(void)
{
}
void Isr_dp_int(void)
{
}
```

7. Main Loop

Once powered, the microprocessor must initialize its ports, memory, timer, and ISR handler. The microprocessor then scans the keyboard and waits for the mode selection from the user. This is important to ensure that the ISP1109 does not operate before the microprocessor is ready to serve it, and that users can check connections and settings before selecting the operation mode.

In the Main Loop routine, the microprocessor polls for any activity on the keyboard. If a key is pressed on the keyboard, the handle key commands execute the routine and then return to the selected Mode Loop. For example:

```
void main(void)
{
   BOOL in_loop = TRUE;
   UCHAR key;
   Initialize();
   Main_Menu();
   /* Main program loop */
   while( in_loop )
   {
      if( bioskey(1) )
      {
         key = bioskey(0);
         key &= 0xff;
         switch(key)
         {
            case 0x1b: /* ESC */
                  in_loop = FALSE;
                  break;
            case 'M':
            case 'm':
               Main_Menu();
               break;
            case 'S':
            case 's':bD09flags.bits.bus_state = SPI;
               Hal4Sys_SetSPI_Mode();
               break;
            case 'I':
            case 'i':bD09flags.bits.bus_state = I2C;
               Hal4Sys_SetI2C_Mode();
               break;
            case 't':
```

```
            case 'T':
               Test_mode();
               break;
            case 'f':
            case 'F':
               Functional_mode();
               break;
            default:
                 break;
            }
         }
   } // Main Loop
   on_exit();
}
```

On reaching the selected Mode Loop, the microprocessor will set a default mode for the ISP1109, and then poll for any activity on the keyboard. If a key is pressed on the keyboard, the handle key commands will execute the routine and set the ISP1109 to return to the selected mode. For example:

```
void Functional_mode(void)
{
    UCHAR key;

   BOOL in_loop = TRUE;
   F_Main_Menu();
   while( in_loop )
   {
      key = 0;
      if( bioskey(1) )
      {
          key = bioskey(0);
          key &= 0xff;
          switch(key)
          {
            case 0x1b: /* ESC */
                  in_loop = FALSE;
                  break;
            case 'm':
            case 'M':
               F_Main_Menu();
               break;
            case 'p':
            case 'P':bD09flags.bits.pin_reg_state = PIN;

Hal4Sys_SpeedSuspend_Pin();

break;
            case 'r':
            case 'R':bD09flags.bits.pin_reg_state = REG;

Hal4Sys_SpeedSuspend_Reg();
```

```
            break;
        case 'l':
        case 'L':bD09flags.bits.low_full_state = LOW;

Hal4Sys_USB_LowSpeed();
            break;
        case 'f':
        case 'F':bD09flags.bits.low_full_state = FULL;

Hal4Sys_USB_FullSpeed();
            break;
        case 's':
        case 'S':bD09flags.bits.suspend_state = SUSPEND;

Hal4Sys_USB_Suspend();
            break;
        case 'a':
        case 'A':bD09flags.bits.suspend_state = ACTIVE;

Hal4Sys_USB_Active();
            break;
        case 'u':
        case 'U':bD09flags.bits.mode_state = USB_VPVM_U;

Hal4Sys_USB_VPVM_U();
            break;
        case 'v':
        case 'V':bD09flags.bits.mode_state = USB_VPVM_B;

Hal4Sys_USB_VPVM_B();
             break;
        case 'd':
        case 'D':bD09flags.bits.mode_state = USB_DAT_SE0_U;

Hal4Sys_USB_DAT_SE0_U();
            break;
        case 'b':
        case 'B':bD09flags.bits.mode_state = USB_DAT_SE0_B;

Hal4Sys_USB_DAT_SE0_B();
            break;
        case '1':bD09flags.bits.mode_state = UART1;

Hal4Sys_UART1();
            break;
        case '2':bD09flags.bits.mode_state = UART2;

Hal4Sys_UART2();
            break;

        case 't':
```

```
            case 'T':bD09flags.bits.mode_state = Audio_Stereo;

Hal4Sys_Stereo();
               break;
            case 'n':
            case 'N':bD09flags.bits.mode_state = Audio_Mono;

Hal4Sys_Mono();
               break;
            case 'h':
            case 'H': bD09flags.value = 0;

Hal4Sys_ResetPCBA();
               break;
            default:
                  break;
         }
      }
   }
    Main_Menu();
}
```

8. References

- ISP1109 Universal Serial Bus transceiver with carkit support data sheet
- CEA-936-A, Mini-USB Analog Carkit Interface.
- Universal Serial Bus Specification Rev. 2.0.

9. Disclaimers

Life support — These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips Semiconductors customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors for any damages resulting from such application.

Right to make changes — Philips Semiconductors reserves the right to make changes in the products - including circuits, standard cells, and/or software - described or contained herein in order to improve design and/or performance. When the product is in full production (status 'Production'), relevant changes will be communicated via a Customer Product/Process Change Notification (CPCN). Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no licence or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

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

10. Trademarks

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

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

11. Contents

Date of release: 17 May 2005

Published in The Netherlands