PAA5101: SPI Interface Reference Control Flow

Application Note AN01

Related Part Ordering Information

Part Number	Type
PAA5101	Optical Tracking Miniature Chip

Lead (Pb) Free
RoHS 6 fully
compliant

For any additional inquiries, please contact us at http://www.pixart.com/contact.asp.

1.0 Introduction

This application note is to describe the control flow of PAA5101 interfacing 3-wire SPI slave with 4-wire SPI master in the host micro-controller. The interfacing circuit and firmware pseudo codes are provided as reference and changes can be made conforming to the SPI specifications and characteristics in the host controller.

2.0 SPI Connection between Host and Slave

The PAA5101 is always being implemented in 3-wire SPI slave mode to interface with the host controller in master mode. Most of the SPI interface support in the host controller is the standard 4-wire SPI master mode. The SDIO signal in the 3-wire SPI, which is the bi-directional serial data input and output signal is to be interconnected with the two serial data signals of MOSI (Master Out Slave In) and MISO (Master In Slave Out) from the host controllers. In this case, the host controller can be connected to the PAA5101 using the connection shown in Figure 1 to have SPI communication with each other.

Note that the R1 resistor of 3.3K ohm is a reference value only. The resistance will have to be determined according to I/O capability of the implemented host controller.

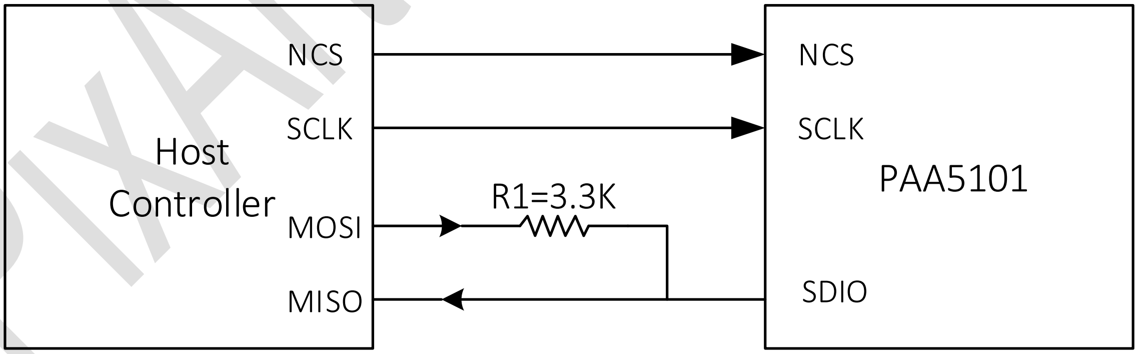

Figure 1. 4-wire SPI Master connected with 3-wire SPI Slave

3.0 Pseudo Codes for the Control of SPI Slave Sensor

```
//==========================================================================================
// The function SPIWrite() and SPIRead() are not shown in this pseudo codes. Users can follow the SPI protocols described
// in the datasheet to program the two functions accordingly.
//==========================================================================================
void main(void)
{
    int16_t deltaX, deltaY;
    bool Init_Success=0;

    Init_Success = Sensor_Init();// Check SPI link and apply sensor settings
    while (Init_Success==1)
    {
        // Check to see if the output buffer is empty
        // Loop in a pre-defined period (e.g. 8ms for USB Low Speed), set Loop_Flag to true while the loop time is up.
        if (Output_Buffer_Is_Empty & Loop_Flag)
        {
                Loop_Flag = 0
                Sensor_ReadMotion(&deltaX,&deltaY);
                if( (deltaX | deltaY) != 0 ) // Check if sensor motion data is available
                {
                        //inverse Y if necessary
                        //deltaY = - deltaY;

                        // Report non-zero X,Y data
                        PutIntoOutputBuffer(deltaX, deltaY);
                }
        }
    }
}
```

```
//=========================================================================================
// Sensor register settings initialization process.
// 1. In SPIWriteRead() function, a write command followed by a read command is to ensure the recommended settings
//    are written correctly.
// 2. Address 0x7F is write-only. A read to address 0x7F will always returns a zero.
//=========================================================================================
bool Sensor_Init(void)
{
        NCS_pin = 1;    // NCS pin keep high
        // Read SensorPID in address 0x00 to check if the SPI link is valid, PID should be 0x31.
        SensorPID = SPIRead(0x00);
        if(SensorPID != 0x31)
        {
                SPI_OK=0;
        }
        else
        {
                SPIWrite(0x06, 0x80);           // chip RESET
                delay_ms(1);                    // short delay, 1ms at least is necessary
                SensorPID = SPIRead(0x00);      // to check if the SPI link is valid after chip reset
                if(SensorPID != 0x31)
                {
                        SPI_OK=0;
                }
                else
                {
                        SPI_OK=1;
                        SPIWriteRead(0x09, 0x5A);       // disable write protect
                        SPIWriteRead(0x51, 0x06);       // To set LD power first, power should be <= 6
                        PAA5101_SETTING_V0P3();         // Load initial settings V0.3
                        SPIWriteRead(0x5D, 0x3E);
                        delay_ms(10);                   // 10ms delay
                        SPIWriteRead(0x5D, 0x3F);
                        PAA5101_LD_MODE();              // LD mode is default
                        SPIWriteRead(0x09, 0x00);       // enable write protect
                }
        }
        return SPI_OK;
}
```

```
//=========================================================================================
// Initial settings for PAA5101
//=========================================================================================
void PAA5101_SETTING_V0P3(void)
{
        SPIWrite(0x7F,0x00);            // Bank0, not allowed to perform SPIWriteRead
        SPIWriteRead(0x05,0xA8 );
        SPIWriteRead(0x07,0xCC );
        SPIWriteRead(0x0A,0x17 );
        SPIWriteRead(0x0D,0x05 );
        SPIWriteRead(0x0E,0x05 );
        SPIWriteRead(0x1B,0x43 );
        SPIWriteRead(0x25,0x2E );
        SPIWriteRead(0x26,0x35 );
        SPIWriteRead(0x2E,0x40 );
        SPIWriteRead(0x32,0x40 );
        SPIWriteRead(0x33,0x02 );
        SPIWriteRead(0x34,0x00 );
        SPIWriteRead(0x36,0xE0 );
        SPIWriteRead(0x3E,0x14 );
        SPIWriteRead(0x44,0x02 );
        SPIWriteRead(0x51,0x06 );
        SPIWriteRead(0x52,0x0C );
        SPIWriteRead(0x57,0x05 );
        SPIWriteRead(0x59,0x03 );
        SPIWriteRead(0x5B,0x04 );
        SPIWriteRead(0x5D,0x3B );
        SPIWriteRead(0x7C,0xC8 );

        SPIWrite(0x7F,0x01);            // Bank1, not allowed to perform SPIWriteRead
        SPIWriteRead(0x00,0x2F );
        SPIWriteRead(0x08,0x1C );
        SPIWriteRead(0x0A,0x02 );
        SPIWriteRead(0x19,0x40 );
        SPIWriteRead(0x1B,0x10 );
        SPIWriteRead(0x1D,0x18 );
        SPIWriteRead(0x1F,0x12 );
        SPIWriteRead(0x20,0x00 );
        SPIWriteRead(0x21,0x80 );
        SPIWriteRead(0x23,0x60 );
        SPIWriteRead(0x25,0x64 );
        SPIWriteRead(0x27,0x64 );
        SPIWriteRead(0x2B,0x78 );
        SPIWriteRead(0x2F,0x78 );
        SPIWriteRead(0x39,0x78 );
        SPIWriteRead(0x3B,0x78 );
        SPIWriteRead(0x3D,0x78 );
        SPIWriteRead(0x3F,0x78 );
        SPIWriteRead(0x44,0x7E );
        SPIWriteRead(0x45,0xF4 );
        SPIWriteRead(0x46,0x01 );
        SPIWriteRead(0x47,0x2C );
        SPIWriteRead(0x49,0x90 );
        SPIWriteRead(0x4A,0x05 );
        SPIWriteRead(0x4B,0xDC );
        SPIWriteRead(0x4C,0x07 );
        SPIWriteRead(0x4D,0x08 );
        SPIWriteRead(0x51,0x02 );
        SPIWriteRead(0x52,0xBC );
        SPIWriteRead(0x53,0x02 );
        SPIWriteRead(0x54,0xBC );
        SPIWriteRead(0x55,0x07 );
        SPIWriteRead(0x56,0x08 );
        SPIWriteRead(0x57,0x07 );
        SPIWriteRead(0x58,0x08 );
        SPIWriteRead(0x59,0x08 );
        SPIWriteRead(0x5A,0x08 );

        SPIWrite(0x7F,0x02);               // Bank2, not allowed to perform SPIWriteRead
        SPIWriteRead(0x07,0x1B );
        SPIWriteRead(0x08,0x1F );
        SPIWriteRead(0x09,0x23 );
        SPIWriteRead(0x51,0x01 );

        SPIWrite(0x7F,0x03);               // Bank3, not allowed to perform SPIWriteRead
        SPIWriteRead(0x07,0x07 );
        SPIWriteRead(0x08,0x06 );
        SPIWriteRead(0x2F,0x00 );
        SPIWriteRead(0x30,0x20 );
        SPIWriteRead(0x32,0x59 );
        SPIWriteRead(0x33,0xD8 );
        SPIWriteRead(0x34,0x4E );
        SPIWriteRead(0x35,0x20 );
        SPIWriteRead(0x36,0x5B );
        SPIWriteRead(0x37,0xCC );
        SPIWriteRead(0x38,0x50 );
        SPIWriteRead(0x39,0x14 );

        SPIWrite(0x7F,0x04);            // Bank4, not allowed to perform SPIWriteRead
        SPIWriteRead(0x05,0x01 );
        SPIWriteRead(0x2C,0x06 );
        SPIWriteRead(0x2E,0x0C );
        SPIWriteRead(0x30,0x0C );
        SPIWriteRead(0x32,0x06 );
        SPIWriteRead(0x34,0x03 );
        SPIWriteRead(0x38,0x17 );
        SPIWriteRead(0x39,0x71 );
        SPIWriteRead(0x3A,0x18 );
        SPIWriteRead(0x3B,0x4D );
        SPIWriteRead(0x3C,0x18 );
        SPIWriteRead(0x3D,0x4D );
        SPIWriteRead(0x3E,0x14 );
        SPIWriteRead(0x3F,0xD1 );
        SPIWriteRead(0x40,0x14 );
        SPIWriteRead(0x41,0xDD );
        SPIWriteRead(0x42,0x0A );
        SPIWriteRead(0x43,0x6C );
        SPIWriteRead(0x44,0x08 );
        SPIWriteRead(0x45,0xAD );
        SPIWriteRead(0x46,0x06 );
        SPIWriteRead(0x47,0xF2 );
        SPIWriteRead(0x48,0x06 );
        SPIWriteRead(0x49,0xEC );
        SPIWriteRead(0x4A,0x06 );
        SPIWriteRead(0x4B,0xEC );
        SPIWriteRead(0x53,0x08 );

        SPIWrite(0x7F,0x05);            // Bank5, not allowed to perform SPIWriteRead
        SPIWriteRead(0x03,0x00 );
        SPIWriteRead(0x09,0x01 );
```

```
        SPIWriteRead(0x0B,0xFF );
        SPIWriteRead(0x0D,0xFF );
        SPIWriteRead(0x0F,0xFF );
        SPIWriteRead(0x11,0xFF );
        SPIWriteRead(0x12,0xD2 );
        SPIWriteRead(0x13,0xD2 );
        SPIWriteRead(0x19,0xFF );
        SPIWriteRead(0x1B,0xFF );
        SPIWriteRead(0x1D,0xFF );
        SPIWriteRead(0x1F,0xFF );
        SPIWriteRead(0x20,0xD2 );
        SPIWriteRead(0x21,0xD2 );
        SPIWriteRead(0x2F,0x7C );
        SPIWriteRead(0x30,0x05 );
        SPIWriteRead(0x41,0x02 );
        SPIWriteRead(0x53,0xFF );
        SPIWriteRead(0x5F,0x02 );

        SPIWrite(0x7F,0x06);            // Bank6, not allowed to perform SPIWriteRead
        SPIWrite(0x2A,0x05 );           // Write ONLY address, not allowed to perform SPIWriteRead
        SPIWriteRead(0x35,0x19 );

        SPIWrite(0x7F,0x07);            // Bank7, not allowed to perform SPIWriteRead
        SPIWriteRead(0x00,0x01 );
        SPIWriteRead(0x14,0x03 );
        SPIWriteRead(0x15,0x14 );
        SPIWriteRead(0x46,0x03 );

        SPIWrite(0x7F,0x00);            // Bank0, not allowed to perform SPIWriteRead
}
```

```
//==========================================================================================
// 1. Read the Motion bit (bit7 in address 0x02) to check if the motion data of X/Y are available to read.
// 2. If Motion bit=1, read X/Y motion data in address 0x03, 0x04, 0x11 and 0x12. Please be noted that the X/Y motion
//    data length are 16-bits (power on default), users can change to 8-bits data length by writing register 0x19 (refer to
//    datasheet).
// 3. The 16-bit X/Y motion data are in 2's compliment format and range from -32768 to +32767.
// 4. It also handles LD and LED switch process.
//==========================================================================================
#define LD2LED_TH       0x700
#define LED2LD_TH       0x500

uint16_t FIQ[8];
uint16_t FIQ_AVG = 0;
uint8_t FIQt = 0;
uint8_t EXTLED_ON = 0;          // Mode index, 0:LD, 1:LED

void PAA5101_LD_MODE(void);
void PAA5101_EXTLED_MODE(void);

void Sensor_ReadMotion(int16_t *dx, int16_t *dy)
{
    int16_t deltaX_l=0, deltaY_l=0;
    int16_t deltaX_h=0, deltaY_h=0;
    uint8_t data_msb, data_lsb = 0;
    uint8_t loopi = 0;
    FIQ_AVG = 0;

    // LD/LED switch process START
    data_msb =  SPIRead (0x75);
    data_lsb =  SPIRead (0x76);
    FIQ[FIQt] = ((uint16_t)(data_msb))*256 + (uint16_t)data_lsb;

    if(FIQt==7)                 //every 8 sampling to decide LD/LED mode
    {
        for(loopi=0;loopi<8;loopi++)
        {
            FIQ_AVG = FIQ_AVG + FIQ[loopi];
        }

        if(EXTLED_ON == 1 && FIQ_AVG < LED2LD_TH) // Check if change to LD MODE
        {
            PAA5101_LD_MODE();
            delay_ms(40);                               // delay for light source change
            SPIWrite(0x03,0x00);
        }
        else if(EXTLED_ON == 0 && FIQ_AVG < LD2LED_TH) // Check if change to external LED MODE
        {
            PAA5101_EXTLED_MODE();
            delay_ms(40);                               // delay for light source change
            SPIWrite(0x03,0x00);
        }
    }
    FIQt = (FIQt+1) & 0x07;
    // LD/LED switch process END

    // Read out delta X/Y motion
    if( SPIRead(0x02) & 0x80 )          //check motion bit in bit7
    {
        deltaX_l = (int16_t)SPIRead(0x03);
        deltaY_l = (int16_t)SPIRead(0x04);
        deltaX_h = ((int16_t)SPIRead(0x11))<<8;
        deltaY_h = ((int16_t)SPIRead(0x12))<<8;
    }

    *dx = deltaX_h | deltaX_l;
    *dy = deltaY_h | deltaY_l;
}
void PAA5101_LD_MODE(void)
{
    EXTLED_ON = 0;                      // Mode index: LD

    SPIWrite(0x7F, 0x00);               // Bank0, not allowed to perform SPIWriteRead
    SPIWriteRead(0x09, 0x5A);           // disable write protect
    SPIWriteRead(0x53, 0x01);
    SPIWriteRead(0x07, 0xCC);
    SPIWriteRead(0x0D, 0x05);
    SPIWriteRead(0x0E, 0x05);
    SPIWriteRead(0x19, 0x24);
    SPIWrite(0x7F, 0x01);               // Bank1, not allowed to perform SPIWriteRead
    SPIWriteRead(0x1D, 0x18);
    SPIWriteRead(0x1F, 0x12);
```

```
        SPIWriteRead(0x42, 0x40);
        SPIWriteRead(0x37, 0x60);
        SPIWriteRead(0x43, 0x0A);
        SPIWrite(0x7F, 0x04);           // Bank4, not allowed to perform SPIWriteRead
        SPIWriteRead(0x06, 0x03);
        SPIWrite(0x7F, 0x05);           // Bank5, not allowed to perform SPIWriteRead
        SPIWriteRead(0x2E, 0x02);
        SPIWriteRead(0x48, 0x00);
        SPIWriteRead(0x3E, 0x05);
        SPIWrite(0x7F, 0x06);           // Bank6, not allowed to perform SPIWriteRead
        SPIWriteRead(0x34, 0x01);
        SPIWrite(0x7F, 0x00);           // Bank0, not allowed to perform SPIWriteRead
        SPIWriteRead(0x09, 0x00);       // enable write protect

        GPIO_LDP_ENL = 0;               // GPIO controls PMOS to low (i.e. turn on LD power)
}
void PAA5101_EXTLED_MODE(void)
{
        EXTLED_ON = 1;                  // Mode index: LED
        GPIO_LDP_ENL = 1;               // GPIO controls PMOS to high (i.e. turn off LD power)

        SPIWrite(0x7F, 0x00);           // Bank0, not allowed to perform SPIWriteRead
        SPIWriteRead(0x09, 0x5A);       // disable write protect
        SPIWriteRead(0x07, 0x55);
        SPIWriteRead(0x0D, 0x7D);
        SPIWriteRead(0x0E, 0x7D);
        SPIWriteRead(0x19, 0x3C);
        SPIWrite(0x7F, 0x01);           // Bank1, not allowed to perform SPIWriteRead
        SPIWriteRead(0x1D, 0x00);
        SPIWriteRead(0x1F, 0x00);
        SPIWriteRead(0x42, 0x20);
        SPIWriteRead(0x37, 0x18);
        SPIWriteRead(0x43, 0x02);
        SPIWrite(0x7F, 0x04);           // Bank4, not allowed to perform SPIWriteRead
        SPIWriteRead(0x06, 0x00);
        SPI_Send(0x7F, 0x05);           // Bank5, not allowed to perform SPIWriteRead
        SPIWriteRead(0x2E, 0x08);
        SPIWriteRead(0x48, 0x02);
        SPIWriteRead(0x3E, 0x85);
        SPI_Send(0x7F, 0x06);           // Bank6, not allowed to perform SPIWriteRead
        SPIWriteRead(0x34, 0x09);
        SPI_Send(0x7F, 0x00);           // Bank0, not allowed to perform SPIWriteRead
        SPIWriteRead(0x53, 0x00);
        SPIWriteRead(0x09, 0x00);       // enable write protect
}
```

```
//===========================================================================================
// Sensor register write then read Command.
// A write command followed by a read command is to ensure the recommended settings are written correctly.
//===========================================================================================
void SPIWriteRead(uint8_t address, uint8_t wdata)
{
        uint8_t read_value;
        do
        {
                SPIWrite(address, wdata);               // Write data to specified address
                read_value = SPIRead(address);          // Read back previous written data
        } while(read_value != wdata);                   // Check if the data is correctly written
        return;
}
```

Document Revision History

Revision Number	Date	Description
0.1	04 Dec. 2017	New creation.
0.2	19 Jan. 2018	Add delay flow and modify the direction
0.3	25 Jan. 2018	Modify the program sequence and some typo