PAT9125EL: I²C/SPI Interface Reference Control Flow

Application Note AN01

Related Part Ordering Information

Part Number	Type
PAT9125EL-TKIT	Miniature Optical Track Sensor with I²C interface
PAT9125EL-TKMT	Miniature Optical Track Sensor with SPI interface

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 PAT9125EL interfacing I²C or 3-wire SPI slave to the host micro-controller. The interfacing circuit and firmware pseudo codes are provided as reference and changes can be made conforming to the I²C or SPI specifications and characteristics in the host controller. Be noticed that 2 different part numbers are associated with the PAT9125EL : PAT9125EL-TKIT is I²C interface and PAT9125EL-TKMT is SPI interface. It is not interchangeable between these two part numbers.

2.0 I²C Connection between Host and Slave (PAT9125EL-TKIT)

The PAT9125EL-TKIT is implemented in I²C slave mode to interface with the host controller in master mode. The bus pull-high resistor R1 and R2 of 5K ohm is a reference value only. The resistance will have to be determined according to I/O capability of the host controller and the I²C bus loading of the whole system.

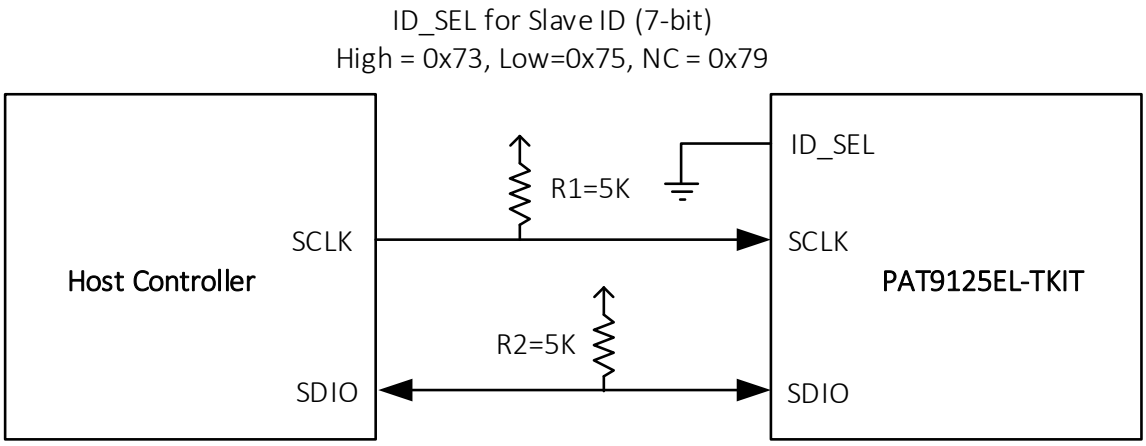

Figure 1. I²C Connecting to Host Controller

3.0 SPI Connection between Host and Slave (PAT9125EL-TKMT)

The PAT9125EL-TKMT is 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 PAT9125EL-TKMT using the connection shown in Figure 2 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.

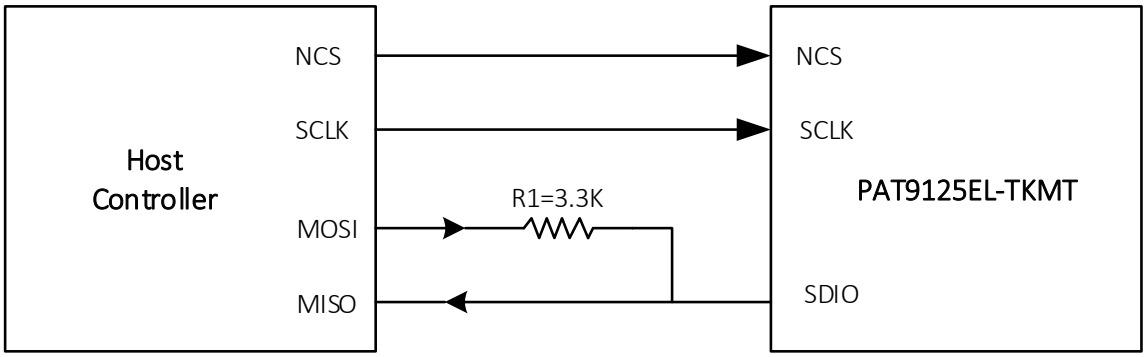

Figure 2. SPI Connecting to Host Controller

4.0 Pseudo Codes for the Control of Sensor

```
//=============================================================================================
// The function OTS_RegWrite() and OTS_RegRead() are not shown in this pseudo codes.
// Users can follow the I2C or SPI protocols described in the datasheet to program the two functions accordingly.
//=============================================================================================
void main(void)
{
    int16_t deltaX, deltaY, dsConutX, dsCountY;
    int32_t sumX=0, sumY=0;
    bool Init_Success=0;

    Init_Success = OTS_Sensor_Init();// Check serial 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;
                OTS_Sensor_ReadMotion(&deltaX, &deltaY);
                if(deltaX || deltaY) // Check if sensor motion data is available
                {
                        //Downscale X and/or Y depends on application
                        dsCountX = OTS_ResDownscale (deltaX, &sumX);
                        //dsCountY = OTS_ResDownscale (deltaY, &sumY);

                        //Output original and/or downscaled data:
                        //PutIntoOutputBuffer (deltaX, deltaY); //deltaX and deltaY are original
                        PutIntoOutputBuffer (dsCountX, CountY); //deltaX is downscaled and deltaY is original
                }
        }
    }
}
```

```
//=============================================================================================
// Sensor register settings initialization process.
// 1. In OTS_RegWriteRead() 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 return a zero.
//=============================================================================================
bool OTS_Sensor_Init(void)
{
        unsigned char sensor_pid=0, read_id_ok=0;

        // Read sensor_pid in address 0x00 to check if the serial link is valid, read value should be 0x31.
        sensor_pid = OTS_RegRead(0x00);
        if(sensor_pid == 0x31)
        {
                read_id_ok =1;

                //PAT9125 sensor recommended settings:
                OTS_RegWrite(0x7F, 0x00);       // switch to bank0, not allowed to perform OTS_RegWriteRead
                OTS_RegWrite(0x06, 0x97);       // software reset (i.e. set bit7 to 1). OTS_RegWriteRead is not allowed
                                                // because this bit will clear to 0 automatically.
                delay_ms(1);                    // delay 1ms
                OTS_RegWrite(0x06, 0x17);       // ensure the sensor has left the reset state.

                OTS_RegWriteRead(0x09, 0x5A);// disable write protect
                OTS_RegWriteRead(0x0D, 0x65);// set X-axis resolution (depends on application)
                OTS_RegWriteRead(0x0E, 0xFF);// set Y-axis resolution (depends on application)
                OTS_RegWriteRead(0x19, 0x04);// set 12-bit X/Y data format (depends on application)
                OTS_RegWriteRead(0x4B, 0x04);// ONLY for VDD=VDDA=1.7~1.9V:  for power saving

                if(OTS_RegRead(0x5E) == 0x04)
                {
                        OTS_RegWriteRead(0x5E, 0x08);
                        if(OTS_RegRead(0x5D) == 0x10)
                                OTS_RegWriteRead(0x5D, 0x19);
                }
                OTS_RegWriteRead(0x09, 0x00);// enable write protect
        }
        return read_id_ok;
}
```

```
//=============================================================================================
// 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 and 0x12.
// 3. The 12-bit X/Y motion data are in 2's compliment format and range from -2048 to +2047
//=============================================================================================
void OTS_Sensor_ReadMotion(int16_t *dx, int16_t *dy)
{
	int16_t deltaX_l=0, deltaY_l=0, deltaXY_h=0;
	int16_t deltaX_h=0, deltaY_h=0;

	if( OTS_RegRead(0x02) & 0x80 ) //check motion bit in bit7
	{
		deltaX_l = OTS_RegRead(0x03);
		deltaY_l = OTS_RegRead(0x04);
		deltaXY_h = OTS_RegRead(0x12);

		deltaX_h = (deltaXY_h << 4) & 0xF00;
		if(deltaX_h & 0x800)  deltaX_h |= 0xf000;

		deltaY_h = (deltaXY_h << 8) & 0xF00;
		if(deltaY_h & 0x800)  deltaY_h |= 0xf000;
	}

	//inverse X and/or Y if necessary
	*dx = -(deltaX_h | deltaX_l);
	*dy = -(deltaY_h | deltaY_l);
}

//=============================================================================================
// This function (Resolution Downscale) provides a more flexible software method for downscaling the resolution
// generated by the hardware built-in method.
// 1. Input data will be accumulated then scaled down.
// 2. Downscale factors (EXPECTED_CNT, ORIGINAL_CNT) should be pre-defined to fit the requirements of the application.
//=============================================================================================
int16_t OTS_ResDownscale(int16_t dCnt, int32_t *sum)
{
	#define EXPECTED_CNT    360
	#define ORIGINAL_CNT    446
	*sum += dCnt;
	return (*sum * EXPECTED_CNT / ORIGINAL_CNT); //return a downscaled value
}
```

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

Document Revision History

Revision Number	Date	Description
0.1	23 May 2016	New creation
0.2	08 Jun 2016	Modified some code statements for setting of Reg0x5D and Reg0x5E.
0.3	07 Oct 2016	Modified software reset procedure in OTS_Sensor_Init(void).