Using the Auxillary Input of the KXPB5

Introduction

This application note explains how to use the auxiliary input of the KXPB5 accelerometer to add an analog sensor input to any application that uses the KXPB5 in SPI mode (mode 00). This feature is useful for adding an analog input to an application where there are no analog inputs available in the main microprocessor, only a SPI bus. As an example, this application note shows a single-axis gyroscope connected to the auxiliary input of the KXPB5. The KXPB5 receives the analog output of the gyroscope and converts it to a 12-bit digital signal that can be read through SPI communication. A recommended circuit schematic and example SPI communication are included.

Circuit Schematics

Figure 1 shows recommended wiring schematics for the KXPB5 when operating in SPI mode, with an Epson XV-3500CB single-axis gyroscope connected to the auxiliary input and a microprocessor connected to the KXPB5 SPI bus. Please refer to the KXPB5 and XV-3500CB data sheets for detailed product and pin descriptions.
http://www.kionix.com/sensors/accelerometer-products.html
http://www.epsontoyocom.co.jp/english/product/Sensor/set01/xv3500cb/index.html

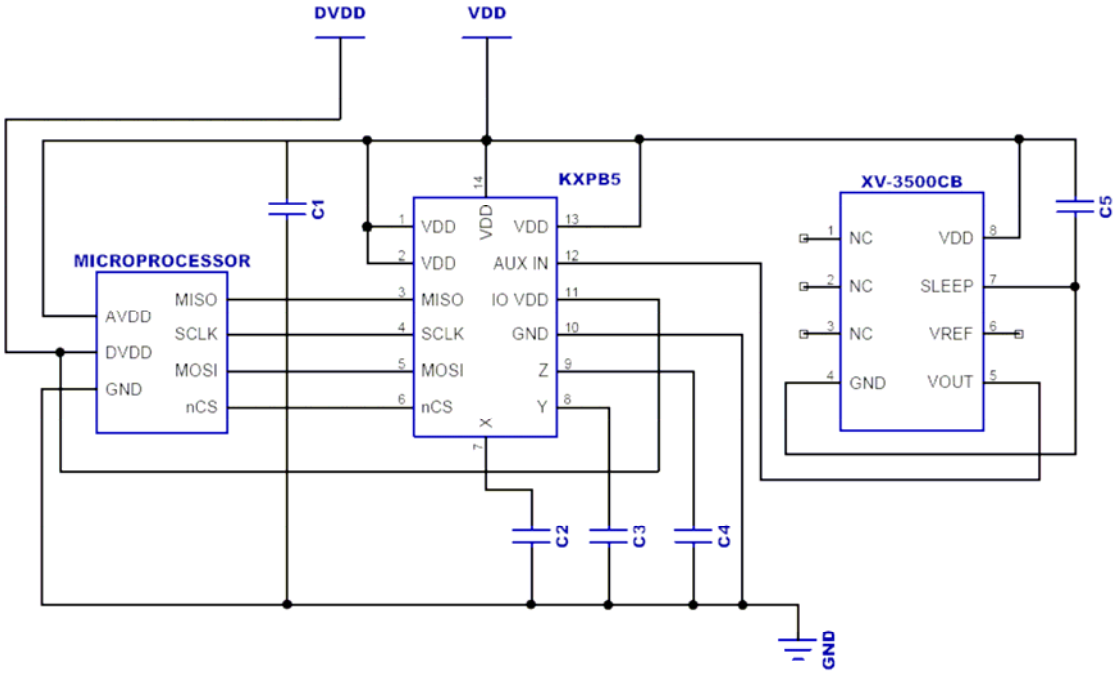

Figure 1. KXPB5 with Epson XV-3500CB gyroscope

Note: these schematics are recommendations based on proven KXPB5 operation. Your specific application may require modifications from these recommendations.

Filter Capacitor Recommendations

To be effective in many applications, such as HDD protection, the KXPB5 needs to respond to changes in acceleration as quickly as possible. The accelerometer's bandwidth, and in turn its response time, is largely determined by the low pass filter frequency response. The KXPB5 has an internal 1kHz low pass filter. For an application needing a lower bandwidth, an external filter capacitance can be added to the accelerometer output pins. The filter capacitance should be as small as the application will allow. Table 1 shows several commonly used bandwidths and the associated capacitor values for C_2, C_3, and C_4 in the circuits shown above. For most applications, 500Hz (0.01μF) should be a good starting point. The noise, and therefore the resolution, of the accelerometer outputs will increase as the bandwidth increases.

Bandwidth (Hz)	Capacitance (μF)
50	0.10
250	0.02
500	0.01

Table 1. Bandwidth (Hz) and Capacitance (μF)

Capacitors C1 and C5, the power supply decoupling capacitors, should have a value of at least 0.1 μF

Converting the auxiliary input

The KXPB5 offers the capability of converting an analog signal present on the auxiliary input to a digital value that can be read through SPI communication. Reading the auxiliary input is very similar to reading the X, Y, or Z acceleration. Table 2 shows the command register addresses that are used to read the X, Y, and Z acceleration and the auxiliary input. Refer to the KXPB5 specifications for more details of the SPI communication protocol and additional command registers.

Register Name	1st byte (SDI) command
X_AXIS	0x00
Y_AXIS	0x01
Z_AXIS	0x02
AUX_IN	0x07

Table 2. KXPB5 Registers

Example firmware for communicating with the KXPB5

Listed below is some example firmware that enables the KXPB5, and then continues to read three acceleration axes and the auxiliary input. A SPI transfer function and a wait function will need to be written for the specific target microcontroller.

```
/* enable KXPB5 by writing 0x04 to the control register */
CS = 0; /* lower chip select */
SPI_Transfer(0x04); /* send 0x04 on the SPI bus to indicate writing to the
control register */
SPI_Transfer(0x04); /* send 0x04 on the SPI bus to enable the KXPB5 */
CS = 1; /* raise chip select */

/* function to convert an axis of the KXPB5 */
unsigned int SPI_Convert_Axis(unsigned char axis)
{
      unsigned int output;
      CS = 0; /* lower chip select */
      SPI_Transfer(axis); /* send convert axis command */

      /* INSERT A FUNCTION HERE TO WAIT 40 MICROSECONDS */

      /* read first byte from KXPB5, left shift it by 1 byte, then read second
      byte from KXPB5 and add it to first byte.  Finally, right shift by 4 bits
      to obtain a 12 bit number. */
      output = ( SPI_Transfer(0x00)<<8 || SPI_Transfer(0x00) )>>4;
      CS = 1; /* raise chip select */
      return output;
}

unsigned int Xaccel;
unsigned int Yaccel;
unsigned int Zaccel;
unsigned int Aux_In;

while(1) /* loop forever */
{
      /* read X acceleration */
      Xaccel = SPI_Convert_Axis(0x00);

      /* read Y acceleration */
      Yccel = SPI_Convert_Axis(0x01);

      /* read Zacceleration */
      Zccel = SPI_Convert_Axis(0x02);

      /* read Auxiliary Input */
      Aux_In = SPI_Convert_Axis(0x07);
}
```