Using the Serial Peripheral Interface of the MLX90609

Application Note 1

MLX90609 AN 1

Using the SPI of the MLX90609

Table of contents

1. Introduction

The MLX90609 has a serial communication interface compatible with the Serial Peripheral Interface (SPI). The serial Interface is used to control an internal ADC, Self-Test logic and to read, write and erase an EEPROM.

This application note includes some suggestions of the hardware interface with the MLX90609 SPI and provides software examples written in C language: one general example and two specific routines written for the microcontrollers from Silicon Labs™ and AVR®. All examples are compatible with the Keil Cx51™ compiler.

2. General description of the SPI

The SPI bus allows high-speed synchronous data exchange between a master device and slave devices. The master is an active part of the bus. It has to provide the clock signal to support a serial data transmission. The slaves can not generate the clock signal and only receive and send data when the clock signal comes from the master. Meaning that the master has to send data to the slave and to read data from the slave in parallel. Signal $\overline{SS}$ (or NSS) is used by the master to choose a particular slave on the bus.

The SPI includes four signal lines:

- serial clock (SCLK or SCK)
- master data output, slave data input (MOSI)
- master data input, slave data output (MISO)
- slave select ($\overline{SS}$ or NSS)

Typically microcontrollers (µC) or Digital Signal Processors (DSP) are used as the SPI-masters to inquire sensors working as the SPI-slaves.

3. MLX90609 SPI data format

The MLX90609 always operates as a slave. Therefore the MISO-pin is an output only.

The MLX90609 SPI is selected when the $\overline{SS}$ pin is low (see Figure 1). When $\overline{SS}$ pin is high, data is not accepted via the MOSI-pin and the serial output pin (MISO) remains in a high impedance state. High level of $\overline{SS}$ forces the serial interface into the start state of data exchange.

Data is serially transmitted to the MLX90609 in 8 bit words (an instruction byte) and in 16 bit data words. Data is serially received from the MLX90609 in 16 bit words (an answer word). Most Significant Bit (MSB) is the first bit transmitted and received.

Transmission:

After the SPI is selected with $\overline{SS}$ going low, the MLX90609 is ready to receive an instruction byte. On each rising edge of SCLK data from the MOSI enter into an internal 8-bit shift register. The instruction byte contains the op-code that defines the operations to be performed.

Reception:

After the 8th bit is received to the shift register, the command is executed by the MLX90609. The format of the outgoing data is defined by the received instruction. Any instruction evokes an answer. A full communication cycle (transmitting an instruction and receiving the full answer) is finished after 24th clock of the SCLK. Communication can be terminated by putting $\overline{SS}$ high.

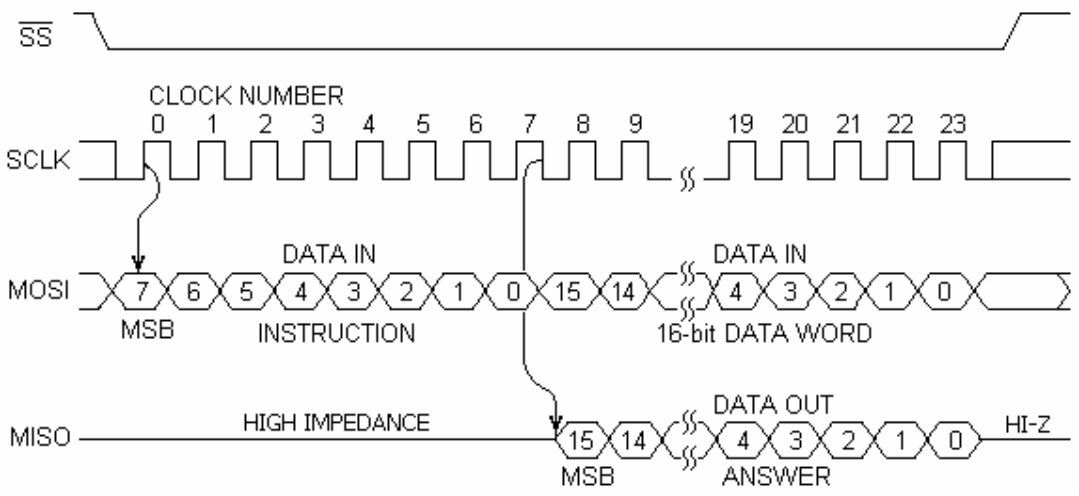

Figure 1. Serial Exchange Sequence (full answer reading)

Some remarks:

- If an invalid op-code is received, it's rejected and the corresponding data is ignored. In this case a special "refusal answer" is generated.
- It's recommended to use synchronization by $\overline{SS}$ after every data exchange to prevent data distortion.
- If high level is applied to $\overline{SS}$ during an instruction byte transmission, the command is ignored.
- If high level is applied to $\overline{SS}$ during the answer reception, the answer will be truncated. It can be used to skip or to limit the reading of the answer to e.g. one byte only (see Figure 2).

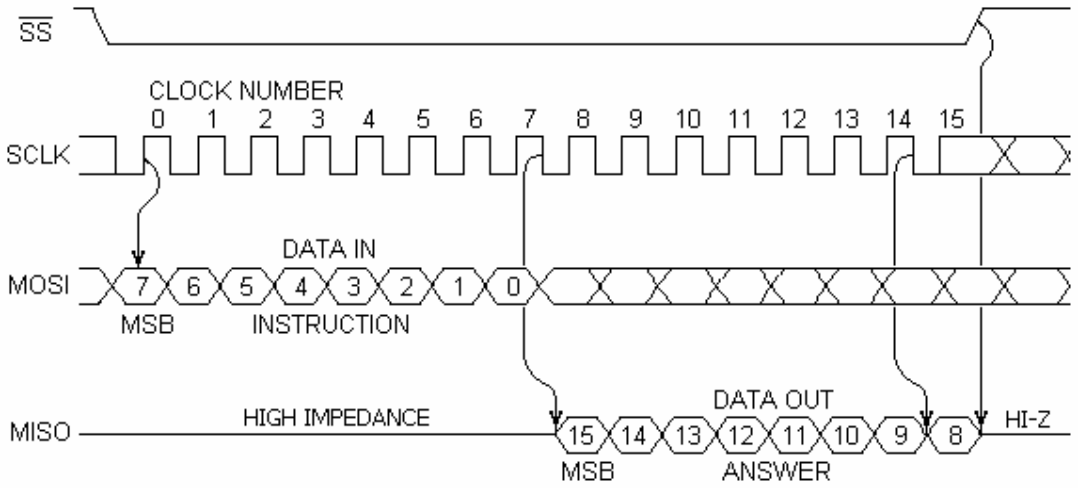

Figure 2. Serial Exchange Sequence (the reading of the higher half of the answer)

4. Hardware interface with the MLX90609 SPI

The MLX90609 was designed for 5-volt digital logic and can be connected to the 5-volt SPI bus directly (see Figure below):

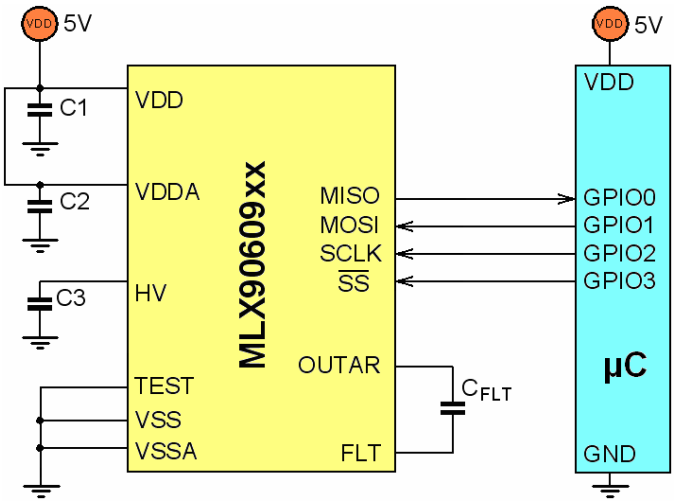

Figure 3: Interface with the 5-volt microcontroller or DSP.

As the SPI bus can include more than one slave, it can be used to combine three MLX90609 sensors working as a 3D gyroscope. Other SPI-slaves (e.g. accelerometers) can share the same bus as well:

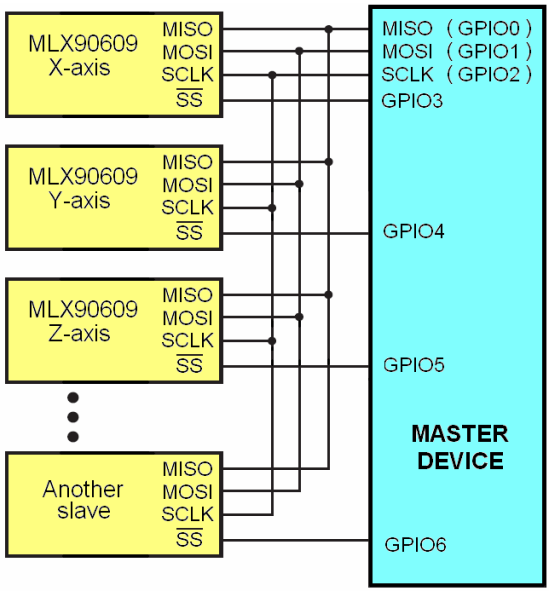

Figure 4: Multi-slave interface with the 5-volt master.

To use the 3-volt microcontroller or DSP, one has to add intermediate buffers for level shifting. Figure 5 shows example of level shifting:

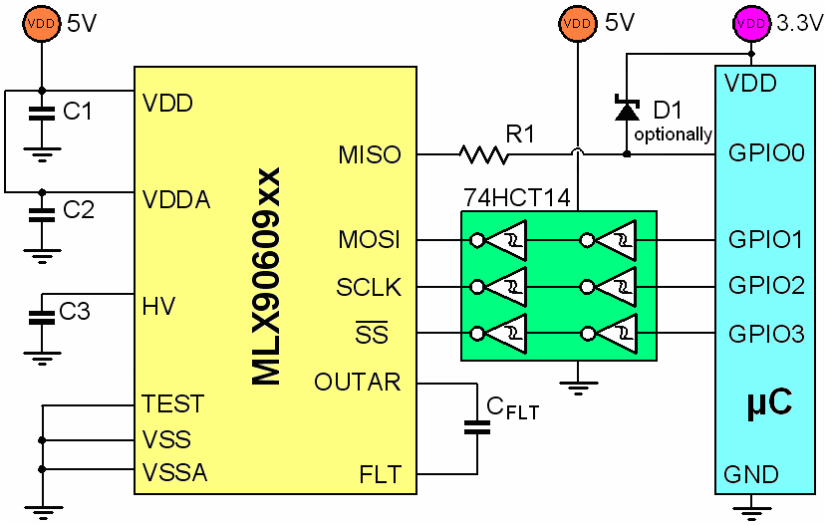

Figure 5: Interface with the 3-volt microcontroller or DSP.

Other elements can be used instead of the shown invertors. On the Figure 6 one can see an example based on the "AND" elements with shorted inputs.

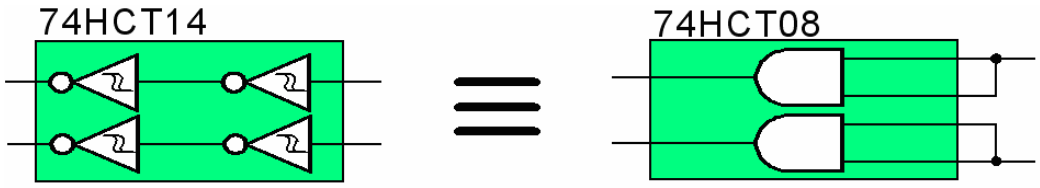

Figure 6: Level shifters on the "AND" elements.

WARNING:

Do not use the logical elements HC instead of HCT, as HC inputs do not guarantee a proper acceptance of the 3-volt outputs.

MLX90609 SPI inputs have internal pull-up and pull-down resistors with a typical resistance of 60kΩ (see Figure below).

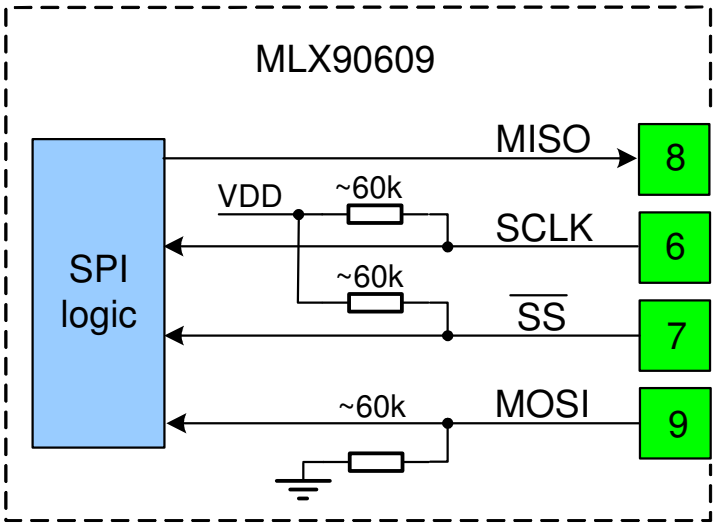

Figure 7: Input circuit of the MLX90609 SPI.

NOTE:

In order to minimize current through the input resistors one can keep inputs SCLK and SS in the high level and MOSI in the low level while SPI is not used. This way approximately 0.25 mA of the total current consumption can is saved.

5. Software Implementation of the MLX90609 SPI

The SPI software implementation can be useful if the master does not have the embedded hardware SPI. For instance some smaller members of the AVR® family do not have hardware SPI.

This section suggests two functions: SPI_EXCH and SPI_FULL. The functions use input-output ports to emulate SPI-signals. As the ways of the ports handling depend on the type of the used microcontroller or DSP, the functions includes six macros to set and to clear MOSI, NSS and SCLK lines and one function Get_MISO to accept MISO-level. The macros and the function should be defined according to the technical description of the used microcontroller. At the end of this section one can find an example of such definition done for the 8051-core microcontrollers.

The SPI_EXCH function is a simplest 8-bit exchange procedure with one input parameter **out**. This is a byte transmitted by the master to the slave (MLX90609). The variable **inp** is used to accumulate bit by bit the input data to be returned at the end of the function. Before return the function recovers the default MOSI-stale (zero level).

Code

```
/* The example code assumes that the part specific header file is included */
/* SCLK_Low/High, MOSI_Low/High and Get_MISO() have to be defined above */
  unsigned char SPI_EXCH(unsigned char out)
{
  unsigned char support, inp;
  for (inp = 0, support = 0x80; support != 0; support >>= 1)
  {
    if(out & support) MOSI_High;    /* MOSI = out & support; */
    else MOSI_Low;     /* It's assumed the delay between SCLK fronts > 80 ns */
    SCLK_Low;                       /* SCLK = 0; */
    SCLK_High;                      /* SCLK = 1; */
    if(Get_MISO()) inp |= support; /* if(MISO) inp |= support; */
  }
  MOSI_Low;                         /* MOSI = 0; */
  return inp;
}
```

The SPI_FULL function uses the SPI_EXCH to provide a full exchange sequence according to the MLX90609 requirements. The input parameters are byte with an instruction **cmd** and 16-bit data word **sdat** that is being shifted into the MLX90609 after the instruction in parallel with answer reading. The function returns the 16-bit MLX90609 answer.

Code

```
/* The example code assumes that the part specific header file is included */
/* NSS_Low and NSS_High have to be defined above */
   unsigned int SPI_FULL(unsigned char cmd, unsigned int sdat)
   {
      unsigned char DatHigh, DatLow;

      DatLow  = sdat;
      DatHigh = sdat>>8;
      NSS_Low;                          /* NSS = 0; */
      SPI_EXCH(cmd);
      /* Necessary delay after EER-instruction only (>= 60us)
      If EER-command is not going to be used, the next line can be removed */
      if(!(cmd & 0x80)) Delay_60us();   /* 0x80 is an EER-instruction */
      DatHigh = SPI_EXCH(DatHigh);
      DatLow  = SPI_EXCH(DatLow);
      NSS_High;                         /* NSS = 1; */
      return (DatHigh<<8) | DatLow;
   }
```

The code below gives an example of the macros' definition written for the 8051-core microcontrollers. As the 8051 architecture supports a bit addressable access to the I/O ports, the macros are trivial:

Code

```
/* The example code assumes that the part specific header file is included */

   sbit SCLK = P1^0;     /* Any available 8051-ports */
   sbit MISO = P1^1;     /* can be used to emulate */
   sbit MOSI = P1^2;     /* the SPI signals: */
   sbit NSS  = P1^3;     /* SCLK, MISO, MOSI and NSS */

   #define NSS_Low   NSS  = 0
   #define NSS_High  NSS  = 1
   #define MISO_Low  MISO = 0
   #define MISO_High MISO = 1
   #define SCLK_Low  SCLK = 0
   #define SCLK_High SCLK = 1

   bit Get_MISO(void)
   {
      return MISO;
   }
```

6. Interfacing the typical SPI of the ATMELs® microcontrollers

The ATMELs® microcontrollers (such as the ATmega™ 48/88/168, ATtiny 48/88, etc.) have a full-duplex SPI that can operate as a master or slave. The data transfer may be set for the LSB first or MSB first (the last choice should be used to interface MLX90609). The AVR's SPI has seven programmable bit rates, end of transmission interrupt flag and collision flag protection. The SPI can wake up the microcontroller from idle mode.

The microcontrollers are able to work with different level of the power supply. Please, take care about the level shifters, if different voltages are used to supply power the MLX90609 and microcontroller.

More information can be found in the ATMEL® datasheets.

It is supposed that the port B supports an embedded SPI:

- PB2 pin is assigned to control the $\overline{SS}$ line
- PB3 pin is assigned to control the MOSI line
- PB4 pin is assigned to control the MISO line
- PB5 pin is assigned to control the SCK line that provides a clock signal for the SCLK input of the MLX90609.

NOTE:

- In the below example the frequency of the SCK is set to be 8 times less than a system clock of the microcontroller. However, another ratio can be set here as well. For example, with a minimal possible ratio 2 the SCK frequency will be 6 MHz for a 12 MHz system clock. This is still in the SPI specification of the MLX90609 (see MLX90609 Data Sheet).
- The SPI_EXCH function described above doesn't depend on the SPI implementation and can be used without changes.

Code

```
/* The example code assumes that the part specific header file is included */

#define NSS_Low   PORTB &= ~(1<<2)
#define NSS_High  PORTB |= 1<<2


// SPI_Init
SPI_Init(void)
{
   DDRB |=    /* Set directions for the SPI lines of the PORTB */
    (1<<2) |  /* Configure PB2=output (NSS)  */
    (1<<3) |  /* Configure PB3=output (MOSI) */
    (1<<5);   /* Configure PB5=output (SCK)  */

   SPSR = (1<<SPI2X);   /* Set SPI double frequency */

   SPCR =        /* Configure SPI mode: */
//  (1<<SPIE) |  /* should be activated to enable interruption from the SPI */
    (1<<SPE)  |  /* to enable SPI */
    (1<<MSTR) |  /* to set Master SPI mode */
    (1<<CPOL) |  /* SCK is high when idle */
    (1<<CPHA) |  /* data is sampled on the trailing edge of the SCK */
//  (1<<SPR1) |  /* In this example  SPI0=1, SPR1=0 (commented) and SPI2X=1 */
    (1<<SPR0);   /* It sets SCK freq. in 8 times less than a system clock */
} /* DORD=0: the MSB is transmitted first */


unsigned char  SPI_EXCH (unsigned char output)
{
   SPDR = output;     /* Start transmission */
   /* Wait till a transmission and reception are completed */
   while(!(SPSR & (1<<SPIF)));
   return SPDR;       /* Return Data Register */
}
```

7. Interfacing the SPI0 of the Silicon Labs™ microcontrollers

The Silicon Labs™ 8-bit mixed signal microcontrollers based on the 8051 core have the Enhanced Serial Peripheral Interface (SPI0) providing access to a flexible, full-duplex synchronous serial bus. The SPI0 can operate as a master or slave device in both 3-wire or 4-wire modes, and supports multiple masters and slaves on a single SPI bus. Additional general purpose port I/O pins can be used to select multiple slave devices in the master mode.

More information can be found in the Silicon Labs™ datasheets.

As the Silicon Labs™ microcontrollers have 3-Volt I/O ports the level shifters recommended in the section 4 should be used to connect the MLX90609 to the SPI0. Note: the microcontrollers' inputs are 5 Volt tolerant. As a result the MISO pins can be connected directly (diode D1 and resistor R1 can be removed, see picture 5).

It is supposed that the GPIO ports are configured and assigned through the Priority Crossbar Decoder of the Silicon Labs™ microcontroller. The SCK line of the SPI0 provides clock signal for the input SCLK of the MLX90609.

Here is an example of the SPI0 configuration. See also the code below.

- Master Mode: Microcontroller operates as a master
- SPI0 Clock Polarity: SCK line high in idle state
- SPI0 Clock Phase: Data centered on the second edge of the SCLK-period
- SPI0 Clock Rate: Let's set it in 4 times less than a system clock of the µC (for example, if the system clock = 24 MHz then SCK frequency will be 6 MHz)

Code

```
/* The example code assumes that the part specific header file is included */
/* SPI_Init */
SPI_Init(void)
{
   SPI0CFG = 0x70; /* SPI Configuration Register */
   SPI0CKR = 0x01; /* SPI Clock Rate Register: Fsck=SYSCLK/(2*(SPI0CKR+1)) */
   SPI0CN  = 0x01; /* SPI Control Register */
}

unsigned char SPI_EXCH(unsigned char out)
{
   SPI0DAT = out;
   while (!SPIF);     /* Wait end of exchange */
   SPI0CN = 0x03;     /* Clear SPIF and other... */
   return SPI0DAT;
}
```

Note: The SPI_EXCH function described above doesn't depend on the SPI implementation and can be used without changes.

8. Disclaimer

Devices sold by Melexis are covered by the warranty and patent indemnification provisions appearing in the Terms of Sale. Melexis makes no warranty, express, statutory, implied, or by description regarding the information set forth herein or regarding the freedom of the described devices from patent infringement. Melexis reserves the right to change specifications and pricing at any time and without notice. Therefore, prior to designing this product into a system, it is necessary to check with Melexis for current information. This product is intended for use in normal commercial applications. Applications requiring extended temperature range, unusual environmental requirements, or high reliability applications, such as military, medical life-support or life-sustaining equipment are specifically not recommended without additional processing by Melexis for each application.

The information furnished by Melexis is believed to be correct and accurate. However, Melexis shall not be liable to recipient or any third party for any damages, including but not limited to personal injury, property damage, loss of profits, loss of use, interruption of business or indirect, incidental or consequential damages, of any kind, in connection with or arising out of the furnishing, performance or use of the technical data herein. No obligation or liability to recipient or any third party shall arise or flow out of Melexis' rendering of technical or other services.

Copyright© 2008 Melexis Microelectronic Systems. All Rights Reserved

Certified ISO/ TS 16949, ISO 14001

For the latest version of this document, visit our website at:

www.melexis.com

For additional information contact Melexis Direct: