Accelerometer CASE

Intelligent instrumentation

Abdullah Özmen
2016-09-25

Summary
Today many electronic devices are in need of an accelerometer to function properly. The accelerometer itself is a small electromechanical sensor that turns the physical acceleration forces into electronic signals. But these signals are often worthless if they are not converted into digital signals, so that is why intelligent instrumentations like the ADXL362 are very useful and popular. These systems are also very popular within embedded systems because they are easy to add and easy to communicate with, trough the SPI interconnection.

Intelligent instrumentation

1.1 Introduction

Today it is very common to find integrated circuits where sensors, data interference and analog-to-digital converters (ADC) are all combined onto a single chip. Such devices are often referred to as *intelligent instrumentations* and are complete are widely used within embedded systems. Standalone sensors have analog output and require the use of ADC to output a digital signal. The main advantage of intelligent systems is the fact that they can be used as modules and can be added to the embedded system using the simple but powerful *serial peripheral interface* (SPI). Embedded systems often have multiple intelligent instrumentations and many of these rely on a few key instruments. Devices like the ADXL362 accelerometer that measures the position and movement of the device is often a very important instrument. That is why this CASE describes the function of the ADXL362 its SPI interconnection.

1.2 Theory

1.2.1 Accelerometer

An accelerometer is a device which can measure acceleration in X-, Y- and Z-axis. The measurements are made in terms of g and the output from the device is analog in nature. An accelerometer is an electromechanical device[1] with various design solutions. Some designs use the capacitor design and has an internal capacitor on every plane of each axis. As the device is moved in space, the acceleration causes the capacitor plates to move thus changing the output in proportion to the acceleration force.

Other accelerometers take advantage of the piezoelectric effect. As seen in figure 1, the piezoelectric effect makes it possible to create designs where the accelerometer contains microscopic crystals structures that get stress by accelerations forces. A small amount of mass attached to a spring is also moved in proportion to the acceleration force and the result is voltage generation.

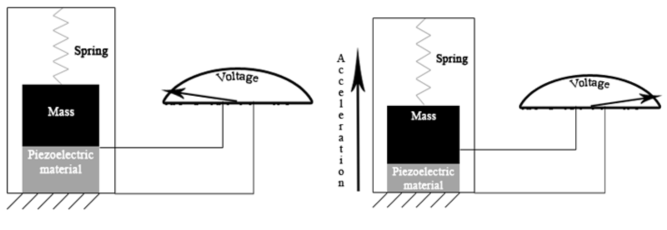


Figure 1: The function of a piezoelectric accelerometer.
Notice the change in the piezoelectric material as the acceleration force change [1].

[1] An Electro Mechanical Device can perform electrical operations using physical moving part. URL: https://en.wikipedia.org/wiki/Electromechanics

The acceleration forces can either be static or dynamic. Static forces are like the gravity, forces that do not change and dynamic forces are forces that change, often caused by movement or vibration. The output of accelerometers is either digital or analog. Analog devices, output continues voltage in proportion to the acceleration while the digital devices uses *pulse width modulation* (PWM). With PWM a square wave is created where the amount of time the voltage is high is in proportion to the acceleration force.

The advantage of an accelerometer is that it helps the device it is used in, to understand its surroundings better. By measuring the amount of static acceleration due to the gravity, the angle of the device can be measured. By measuring dynamic acceleration, the movement of the device can be analyzed. By doing this the device can understand if it goes uphill, if its falling, if it is in horizontal position or if it is tilted. The properties of accelerometers are many. The number of axes if course important, for 3D positioning, 3-axis are necessary. Higher sensitivity of the accelerometer gives a lager signal thus making it easier to measure. The bandwidth and sampling rate is also important and normally a 50Hz sampling rate is sufficient for most applications.

1.2.2 Serial Peripheral Interface (SPI)

1.2.2.1 History

SPI stands for *Serial Peripheral Interface* and is a simple, effective and very popular serial interface. SPI was first intended for use with microcontrollers and was developed by Motorola and National Semiconductors. The goal was to develop a communication standard that would be simple and easy for other manufacturers to interfere with. Motorola called its version for *SPI* and the National Semiconductors for *Microwave*.

In data busses data is transferred parallel, that means that there is one wire for each bit of data and some for synchronization and control. Because data is transferred a word[2] at a time, lots of wires are needed for connection between interconnected devices. To avoid this problem, the effective and simple serial communication was invented, where a single wire can be used to transfer and receive data. However this requires the data to be sent in turn which creates the problem of knowing where each word end or begins. The simple solution was to send the data along a clock signal, synchronizing the serial communication as can be seen in figure 2.

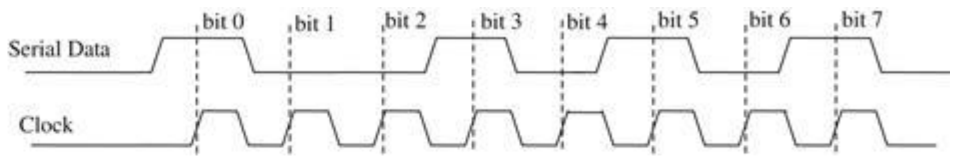


Figure 2: Synchronous data communication example. 1-bit of data is transferred with each clock cycle [2].

[2] A word is a natural unit of data used by the processor. In an 8-bit system the word length is 8 bits or 1 byte.

In figure 2, the synchronization enables 1 bit of data to be sent from the master to the slave and from the slave to the master with each clock cycle. Because data is transmitted in both directions it is called *full duplex communication*.

1.2.2.2 Architecture

SPI standard is used for data communication between two or more devices, where one of these is the *Master Communication Unit* (MCU) that controls all activity on the serial interconnection. In the serial data interconnection devices are called *nodes* and the connection between nodes are called *SPI links,* where the minimum amount of links, is between one master and one slave. If there are more than one connected slave device the *Slave Select* (SS) or *Chip Select* (CS) signal is used to select which slave is to receive the data. As seen be seen in figure 3, the master can communicate with one or more slaves. The slaves are connected to multiple *Parallel Input Output* (PIO) pins on the MCU. As the MCU selects a certain slave the connected SS is driven low. The unselected slaves do not receive the clocked data thus not interfering in data transmission. In the figure 4 the SPI architecture is shown with only one slave. As there is no need to select between slaves the SS is driven low by grounding the signal.

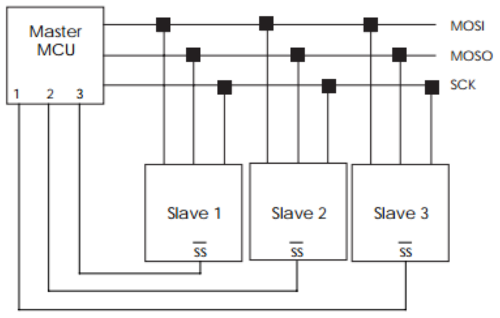


Figure 3: System architecture where the master is connected to more than one slave [2].

Master Out Slave In (MOSI) is the serial data output of the Master and is connected to the serial data input on the slave. *Master In Slave Out* (MISO) is the serial data output of the slave and is connected to the serial data input of the master. With every clock cycle 1-bit of data is transmitted from master to slave and slave to master, thus in an 8-bit system, 1 byte is transferred after 8 clock cycles.

Authors Comment: In our case the circuitry for the master in the system is contained within the FPGA. The slaves that are connected to the master are from the outside world, for example an ADC or an accelerometer.

1.2.2.3 Shift register and Data register

Shift registers in both the master and the slave are one of the most important features of the serial link. It consists of multiple digital flip flops, where each one can hold 1 bit of data. The output of one shift register *Most Significant Bit* (MSB) is connected to the next shift registers *Least Significant Bit* (LSB), as can be seen in figure 4.

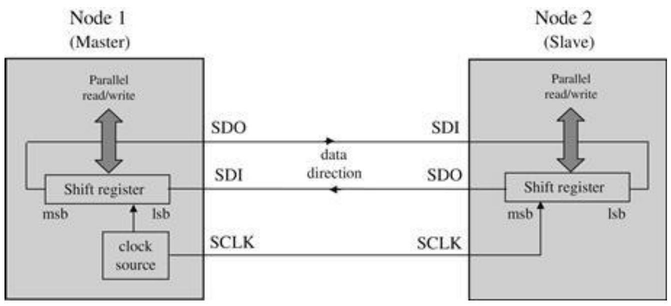


Figure 4: Description of shift register function [4].

In figure 4, data in one shift register MSB, is transmitted to the LSB of the other shift register with every clock cycle. This process continues until the shift registers on both sides are full thus containing a word. The data in the shift registers can be read all at the same time, as a parallel word or a new value can be loaded in. That is why the shift register is considered as a very useful subsystem, because it can convert serial data to parallel data or the other way around. It can also act as a serial transmitter and/or a serial receiver.

The data received to the master and slave shift registers is clocked out to the *Serial Peripheral Interface Data Register* (SPDR) using the common clock signal, see figure 5. In an 8-bit system, when 8 bits have been transferred, an interrupt is locally generated so that the data can be read before the next byte is clocked through.

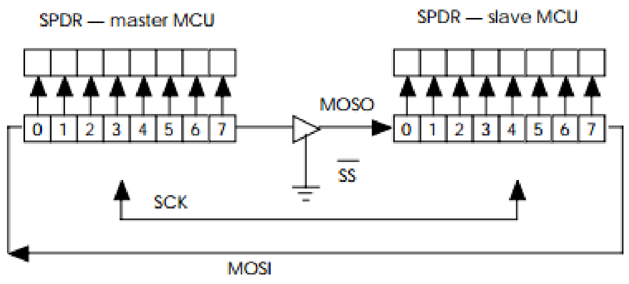


Figure 5: Data in shift registers are transferred to data registers with every clock cycle [2].

The SPI standard does not implement data buffering, neither on the master nor on the slaves. As soon as the master writes the data into the SPDR it is transmitted to the slave. When all the data has been clocked out or transferred, an interrupt is generated indicating that the byte has been transferred and the *Serial Peripheral Interrupt flag* (SPIF) in the *status register* (SPSR) is set. The Master cannot send data until this flag is cleared and must be cleared by an *Interrupt Service Routine* (ISR). All slaves have the same function and do not use data buffering. As data is received to the shift register an interrupt is generated by the status register when a byte is received. The received data is then transferred to the SPDR. It is therefore essential that the interrupt that is generated by the full shift register is serviced quickly by transferring the data before the next byte is transmitted and transferred to the SPDR. The number of clock cycles for the slave/master to receive/transfer the data is effectively determined as the maximum data rate.

Authors comment: The maximum data rate is determined by the internal clock of the MCU. For a given clock of

50MHz the maximum data rate is 50 Mbit/s, that means theoretically 1 bit of data for every clock cycle. Increasing the internal clock speed will thus increase the maximum data rate.

1.2.3 ADXL362

The ADXL362, made by Analog Devices, is an intelligent sensor. This sensor is also called a *Micro Electro Mechanical System* (MEMS) because the accelerometer mechanics are fabricated within the *Integrated Circuit* (IC) structure. The Capacitors fixed to moving mass are deflected and unbalanced in proportion to the acceleration force which results in a proportional acceleration output. Besides this phase sensitive demodulations[3] determines magnitude and polarity of acceleration.

1.2.3.1 Architecture

The ADXL362 is a complete 3-axis acceleration measurement device and can operate at extremely low power consumptions, only 2nA at 100Hz *Output Data Rate* (ODR). It can measure both dynamic acceleration, like motion or shock and static acceleration, like tilting. The output is digital via the SPI interconnection and has a maximum of 12-bit output resolution. It can take measurements in the range of ±2, ±4 and ±8 gees where ±2 is standard. The architecture can be seen in figure 6 below, it has several built in applications like *First In First Out* (FIFO) buffer, SPI output, internal ADC, temperature sensor and accelerometer sensor, which makes it a very useful and versatile intelligent instrument.

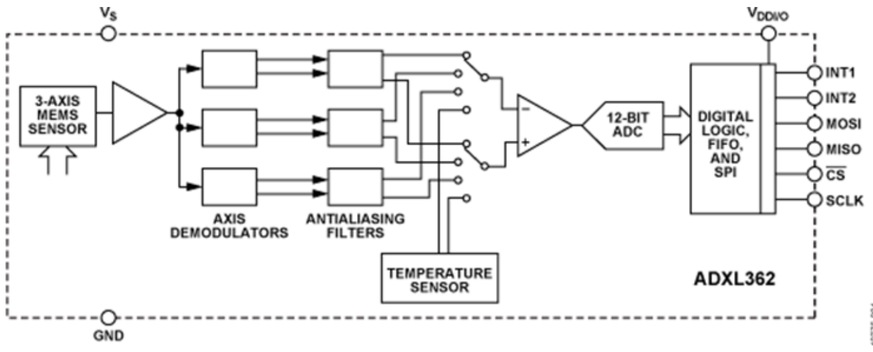


Figure 6: The architecture of ADXL362 [4].

1.2.3.2 Operation Modes

The ADXL362 has two different operation modes, *measurement mode*, for continues sensing and *wake-up mode* for activity/inactivity detection. Beside these, the device can be placed in *standby mode,* suspending all measurement. Measurement mode is the normal operation mode. In this mode, acceleration data is read continually and output data can be in various data rates form 12.5Hz to 400Hz but standard ODR is set to 100Hz.

Wake-up mode can be configured manually. In this mode, acceleration data is measured about 6 times each second, making it possible for the device to detect sudden motions, for example if the device is dropped. This mode is ideal when the device is used as a motion activated on/off switch. This enables the rest of the system to be powered down until activity is detected, thus

[3] Read more about PSD, URL: https://www.google.com/patents/US2562912?hl=sv

eliminating significant system standby power at system level. As motion is sensed the device is able to switch into full measurement mode, send an interrupt signal to the processor or do other preconfigured tasks.

1.2.3.3 Activity/Inactivity detection

In wake-up mode the ADXL362 is able to detect *Activity* or *Inactivity*. When activity or inactivity is detected the event is indicated in the status register with the *AWAKE bit*, se figure 7. This describes the status of the device, if there is presence of dynamic or static acceleration, and is used to perform pre configured tasks. The AWAKE bit can be mapped to the INT1 or INT2 pins (see figure 6) thus allowing the pin to serve as a status output. In figure 7 it can ebe seen that the AWAKE bit has read only access and when the bit is logic high (1), the accelerometer is in an active state. When the bit is logic low (0), the accelerometer is in an inactive state.

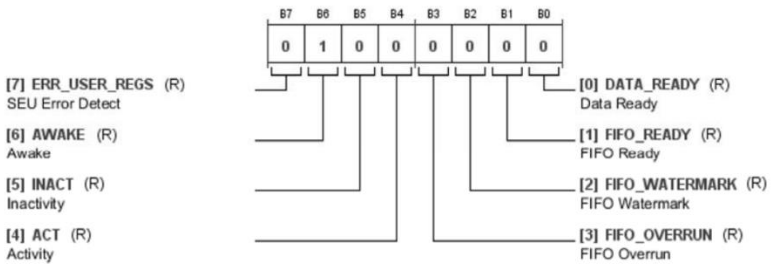


Figure 7: Status registers with address 0x0B [4].

Activity is detected when acceleration remains above a specific threshold for a specific time period, bit 4 in status register is to logic high(1), see figure 7. Inactivity is detected when acceleration is below a specific threshold for a specific time period, bit 5 in status register is to logic high(1), see figure 7. The activity/inactivity time is set by an internal timer that can be set from 2.5ms to almost 90 minutes. It’s important to notice that inactivity measurement is done after device has waited for the specified time period. If the activity/inactivity detection timer is set to 10 minutes, detection is done when the device has been stationary for 10 minutes.

Activity/inactivity can be measured as *reference* or *absolute*. In absolute detection, measurements are compared to a user set threshold for a pre-defined period of time. In reference detection the measurements are compared to a user set threshold from a pre-defined reference (starting point) for a pre-defined period of time. If the user sets threshold at 0.5g and the current acceleration at the x-axis is 0.8 g longer than the pre-configured time period, the activity status asserts.

Authors comment: Absolute measurement should be used when the pre-defined threshold is above 1g. This is because the accelerometer will normally output 1g of static acceleration due to the earth’s gravity.

1.2.3.4 FIFO buffer

The ADXL362 has a 512-sample FIFO buffer enabling the device to store up to 13 seconds of event context/data. The FIFO saves power at system level by enabling the host processor to do other tasks or sleep, while sampling data. The FIFO can also be used for data recording before an event, by sampling data until activity/inactivity is detected. When activity/inactivity is detected, data prior to the event can be sent to the host processor. In general, the more context available for analyze, the more intelligent decisions can be made.

Authors comment: The Airbag in a car is often a life saving equipment. The deployment of an Airbag is relied on the proper function of an accelerometer, and wrong deployment can cause serious damage. The FIFO buffer plays an important role in this. If the accelerometer is activated, the context that caused the activation can be analyzed and decision can be if deployment is necessary.

1.2.3.5 SPI interconnection

Because measurements are converted to 12 bit values by the ADC, this requires two registers to be read on each axis, thus to read all 3-axis, six registers have to be read. However, 8-bit accuracy is often sufficient for many systems, so that is why the MSB registers XDATA, YDATA, and ZDATA can be read. Because only 3 registers have to be read instead, SPI bus activity time is reduced, reducing total power consumption level.

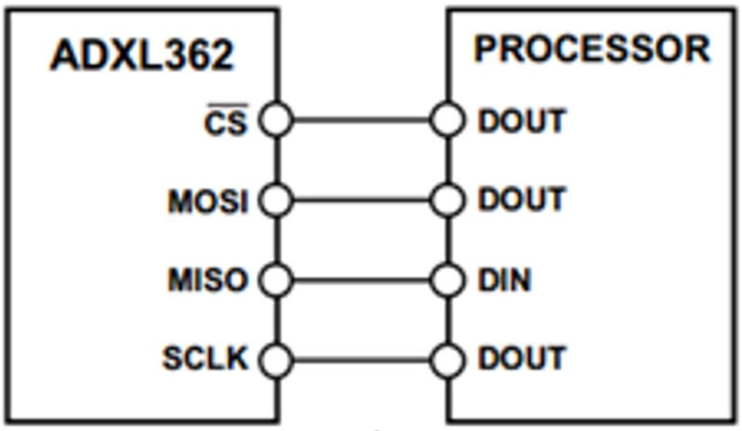


Figure 8: The physical connection of the ADXL362 as a slave connected to the master [4].

The SPI on the ADXL362 communicates with 4 wires that can be seen in figure 8. Because the ADXL362 is operating as a slave, when read data is sent to the master the MISO pin is set to logic low (0).

1.2.3.6 SPI commands

The SPI port on the ADXL362 supports a multibyte system that has following command set,

write register: 0x0A *read register: 0x0,* *read FIFO: 0x0D*

By using multibyte transfer concurrent and complete set of x-, y- and z-axis acceleration data can be read. In figure 9 and 10 the command structures for register read and write is illustrated.

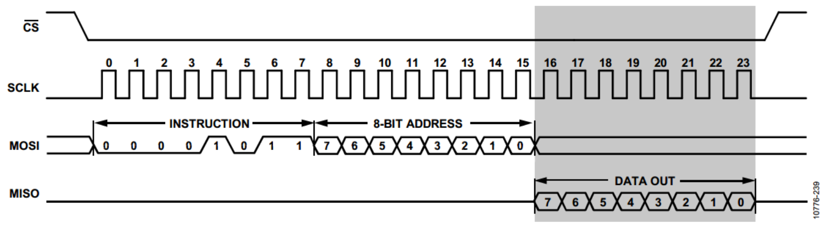


Figure 9: SPI register read.

When reading data from the ADXL362 registers, the command structure in figure 9 should be followed. The instruction *read register* (0x0B) should be followed by an 8-bit address. The SPI will then output 8-bit *read data* from that address.

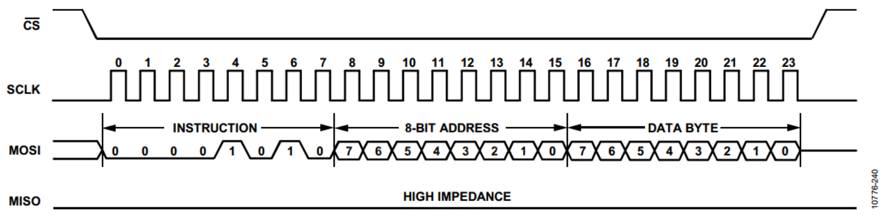


Figure 10: SPI register write. Notice that MISO wire is always logic high when writing data.

When writing data to the ADXL362 registers, the command structure in figure 10 should be followed. The instruction *write register* (0x0A) should be followed by an 8-bit address and then the 8-bit data to be written to that address. Worth noticing is that the ADXL362 does implement read/write auto-increment. The address specified in the read/write process is incremented for each additional data byte. The auto-increment function will continue depending on the size of *write_length* and *read_length*.

Authors comment: If *read_length* is set to 3, this effectively means that *read data* is expected to be 3 bytes and the address is incremented 3 times. For example if start address is at 0x08 on the first byte, it will increment to 0x09 on the second byte and 0x0A on the third byte.

In our CASE the command for SPI communication is as follows:

alt_avalon_spi_command *(base, slave, write_length, write_data, read_length, read_data, flags).*

In table 1 the parameters of the *altera_avalon_spi_command* is described.

Table 1: Description of SPI command parameters.

Name	**Description**
Base	Base address for the accelerometer.
Slave	Slave select or Chip select, use if more than one slave is connected.
Write_length	Write data length.
Write_data	The data to be written to the slave registers.
Read_length	The read data length.
Read_data	The data being sent from the slave device.
Flags	Enable flags, 0 if flags are disabled and 1 if flags are enabled.

1.2.3.7 Communicate with the accelerometer

To start communication with the accelerometer the device has to be initiated. This is done by writing to the *power control register* which is an 8-bit register at address 0x2D. The power control register controls the mode of the accelerometer. By default the device is at standby mode but by writing 0x02 to this register, it sets the device into full measurement mode. When the device has been initiated it will continually output data at 100Hz ODR with the measurement range of ±2g. This configuration is standard but can be altered by writing to the *filter control register* at address 0x2C. To learn more about this please refer to the ADXL362 datasheet [4]. To start reading data from the device simply use the *altera_avalon_spi_command*.

1.3 CASE – Accelerometer instrument

1.3.1 Specifications

In this CASE, 8-bit acceleration data from measurements in X-, Y- and Z-axis, will be displayed on the console. As a result, both static acceleration, like gravity and tilting, and dynamic acceleration, like motion, will be demonstrated. Because the BeMicro board is not sufficiently protected the activity/inactivity detection system will not be demonstrated, due to possible damage to the board.

1.3.2 Hardware

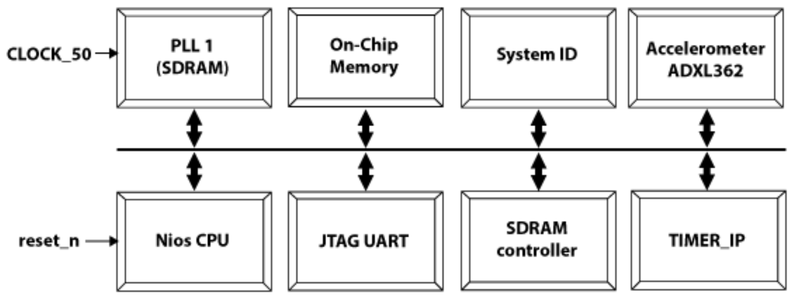


Figure 11: Computer architecture of this CASE

The hardware used in this CASE is illustrated in figure 9. Because the ADXL362 has built in 12-bit ADC, there is no need for additional ADC. However for the purpose of demonstration, an additional Timer component will be used to control the sampling rate outputted to the Nios II console window. P

1.3.3 Implementation

Follow these steps:

1. Create a new project folder and name it Accelerometer_CASE.

2. Unzip the hardware files in into the project folder. It should look as in figure 12.

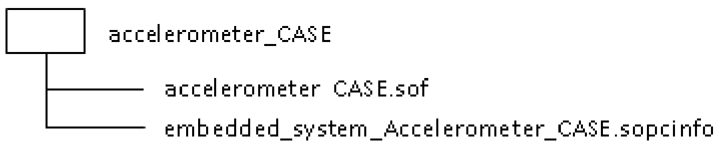


Figure 12: Project folder for accelerometer CASE.

3. Create a new Nios II project named accelerometer_CASE based on the *.sopcinfo* file in figure 12 and the *Hello_world* template. Open the BSP editor and uncheck the *"Support for C++"* option and check the *"Use reduced device drivers"* and *"Use small C library"* options.

4. Open the *Nios II* programmer and download the hardware *(.sof)* to the FPGA.

5. Open the *hello_world.c* source file and enter the following code:

```
/*********************************************************************************
Company: TEIS AB
Engineer: Abdullah Özmen
Create Date: 2016 Sept 15
Design Name: accelerator_CASE.c
Target Devices: BeMicro Max 10
Tool versions: Nios II.
Description:
This is a program to demonstrate the function of the ADXL362 MEMS accelerator attached to the
BeMicro Max 10 board. The program start initiates and starts measurement mode. Measurements
are read every 0.2 seconds and printed to console. By default measurement range is set to +- 2g
and data printed to console is in mg/MSB format. Please refer to accelerator_CASE document for
further description.
Verified with "Nios II" and validated with
 BeMicro Max 10 board by Abdullah Özmen.
*********************************************************************************/
#include <stdio.h>
#include <io.h>
#include <system.h>
#include <alt_types.h>
#include "altera_avalon_timer_regs.h"
#include <altera_avalon_spi_regs.h>
#include "altera_avalon_spi.h"

#define WRITE_COMMAND 0x0A //SPI write command.
#define READ_COMMAND 0x0B //SPI read command.
#define POWER_CONTROL_REGISTER 0x2D
#define MEASUREMENT_MODE 0x02
#define X_AXIS_MSB_REGISTER 0x08

int main()
{
        TIMER_RESET; //Reset timer.
        TIMER_START; //Start timer.

        //Variables
        alt_u8 spi_command_tx[2];
        alt_u8 spi_command_rx[3];
        typedef struct mystruct {
                alt_8 x;
                alt_8 y;
                alt_8 z;
        } ACCELEROMETER;
        ACCELEROMETER accel_data;

        /* START ACCELEROMETER MEASUREMENT MODE */
        spi_command_tx[0] = WRITE_COMMAND; // write command
        spi_command_tx[1] = POWER_CONTROL_REGISTER; // Power control register
```

```
		spi_command_tx[2] = MEASUREMENT_MODE; // Measurement mode/start.

		//SPI command.
		alt_avalon_spi_command(ACCELEROMETER_SPI_BASE,0,3,spi_command_tx,0,spi_command_rx,0);

		/* START READING FROM ACCELEROMETER VIA SPI INTERCONNECTION */
		spi_command_tx[0] = READ_COMMAND; // read command
		spi_command_tx[1] = X_AXIS_MSB_REGISTER; // Start reading from X-Axis MSB register.

		while(1){
			if(TIMER_READ > 5000000){ //Run every 0.2 seconds.
				TIMER_STOP; //Stop timer.

				//SPI command.
				alt_avalon_spi_command( ACCELEROMETER_SPI_BASE, 0,2, spi_command_tx,3, &accel_data,0);


				//Print to console.
				printf("X-Axis: %d\t Y-Axis: %d\t Z-Axis: %d\n",accel_data.x,accel_data.y,accel_data.z);

				TIMER_RESET; //Reset timer.
				TIMER_START; //Start timer again
			}
		}
		return 0;
	}
```

6. Build and download the program. Observe in the Nios II console window, data output from the accelerometer. Start with the accelerometer in its top position and notice how the output data is changed as you move and rotate the device according to figure 13.

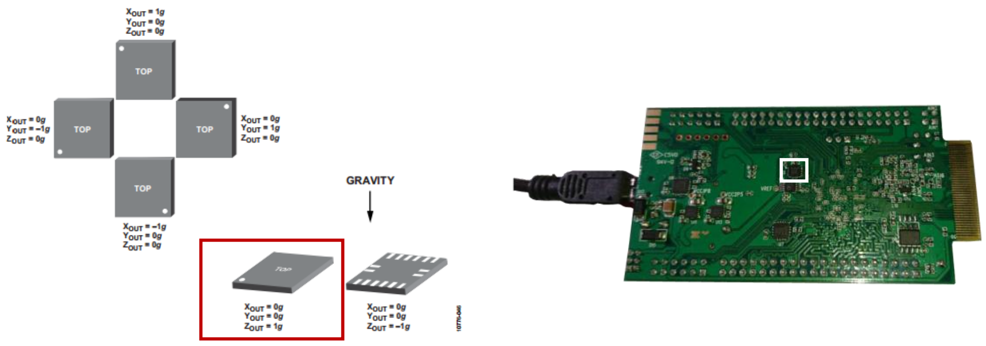
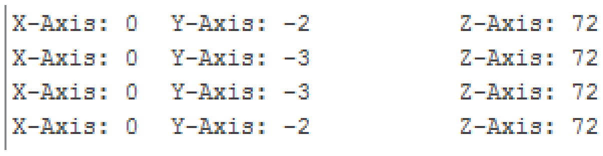


Figure 13: Axis data output according to accelerometer orientation [4].

As the BeMicro board is moved according to description in figure 13, the data output changes. Notice in figure 14 that the data output represents the accelerometer in its horizontal position, top side up, see figure 13.



```
X-Axis: 0  Y-Axis: -2          Z-Axis: 72
X-Axis: 0  Y-Axis: -3          Z-Axis: 72
X-Axis: 0  Y-Axis: -3          Z-Axis: 72
X-Axis: 0  Y-Axis: -2          Z-Axis: 72
```

Figure 14: Data output in mg/MSB for X-, Y- and Z-axis.

The outputted acceleration data in figure 14 is actually an 8-bit value in decimal form. To really understand the output in gees, this is how the conversion is done:

Our measurement range is ±2g which effectively gives us the range of 4g. The outputted signed 8-bit acceleration data gives us in total 256 measurement bins. Thus represents each bit the output value: 4/256 = 0.015625g or 15.625mg. With this knowledge the output data in figure 14 can interpreted to:

```
X-Axis: 0g   Y-Axis: -0.03125g     Z-Axis: 1.125g
X-Axis: 0g   Y-Axis: -0.046875g    Z-Axis: 1.125g
X-Axis: 0g   Y-Axis: -0.046875g    Z-Axis: 1.125g
X-Axis: 0g   Y-Axis: -0.03125g     Z-Axis: 1.125g
```

1.4 Discussion

The SPI standard is extremely effective, simple and cheap. Data is transferred rapidly but there is no acknowledgment from the receiver. This means that the master cannot be sure if the data has been received. Electromagnetic interference can also cause damage to data or clock signal, so SPI may not be suitable for complex and high-reliability systems. However the ADXL362 is a simple but effective device that works well with the SPI interconnection. The accuracy of the outputted acceleration data is as expected but noise from outside world can cause some deflection to the measurements.

1.5 Reference

Literature:

1. R. Tulson & T. Wilmshurst (2012), *Fast And Effective Embedded Systems Design,* Elsevier Science Ltd, Oxford GB, (ISBN: 9780080977683).
2. S. Health (2003), *Embedded Systems Design,* Elsevier Science Ltd, Oxford GB, (ISBN: 07506 55461\ISBN-13: 978-0750655460).

Internet:

3. Dimension Engineering, *A beginner's guide to accelerometers,* http://www.dimensionengineering.com /info/accelerometers (downloaded: 2016-08-25).
4. Analog Devices, *ADXL362 Data sheet,* http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL362.pdf (downloaded: 2016-08-26).

Illustration:

[1]. https://cdn.sparkfun.com/assets/a/9/1/1/7/516daf84ce395f411e000001.gif
[2]. http://read.pudn.com/downloads158/ebook/707037/Embedded%20Systems%20 Design%20-%202ed%20-%200750655461.pdf.
[3]. Fast And Effective Embedded Systems Design
[4]. http://www.analog.com/en/products/mems/accelerometers/adxl362.html?doc= ADXL362.pdf#product-overview

1.6 YouTube recommendations

1. https://www.youtube.com/watch?v=i2U49usFo10.