Accelerometer SPI Mode Core for DE-Series Boards

For Quartus Prime 16.0

This document describes the Altera University Program's IP core that can be used to access the accelerometer peripheral found on Altera's DE0-Nano and VEEK-MT boards.

1 Accelerometer Chip

The accelerometer peripheral consists of the Analog Devices' ADXL345 chip. It measures acceleration in three directions, which are referred to as x-axis, y-axis and z-axis. When stationary, the only acceleration experienced by the chip is due to gravity. The measured results have 13-bit resolution in the range up to ± 16 g.

The accelerometer chip includes a set of eight-bit registers that store the results of measurements and provide settings for various modes of operation. To accommodate the 13-bit size, the measured values of x, y and z are stored in pairs of registers.

The AXDL345 chip supports serial communication using either the SPI (Serial Peripheral Interface Bus) or I^2C (Inter-Integrated Circuit) Standards. The SPI communication is supported using either a three-wire or four-wire connection. On the DE0-Nano and VEEK-MT boards, the accelerometer chip is connected to the FPGA chip by the three-wire SPI arrangement.

A full description can be found in Analog Devices datasheets: *Digital Accelerometer ADXL345*.

2 UP Accelerometer Core Overview

The UP Accelerometer Core provides a convenient interface for accessing the accelerometer chip. The core can be instantiated in systems designed by using the *Qsys* tool in Quartus II software. The core is also included in the DE0-Nano Computer, which can be conveniently used within Altera University Program's Monitor Program.

Figure 1 shows a high-level block diagram of the core. To set up the accelerometer device, the core takes information from a program (via the Avalon bus) or an auto-initialization circuit and sends it out via the serial bus.

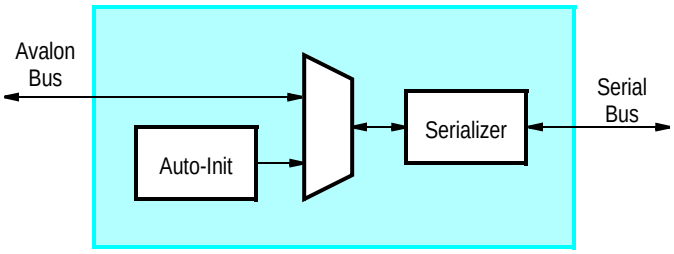

Figure 1. High-level block diagram of the UP Accelerometer Core.

3 Instantiating the Core

The UP Accelerometer core can be instantiated in a system using Qsys or as a standalone component from the IP Catalog within the Quartus II software. Designers who wish to include the UP Accelerometer Core in their custom systems should use the core's configuration wizard. The wizard is depicted in Figure 2. It has no options, as indicated in the figure.

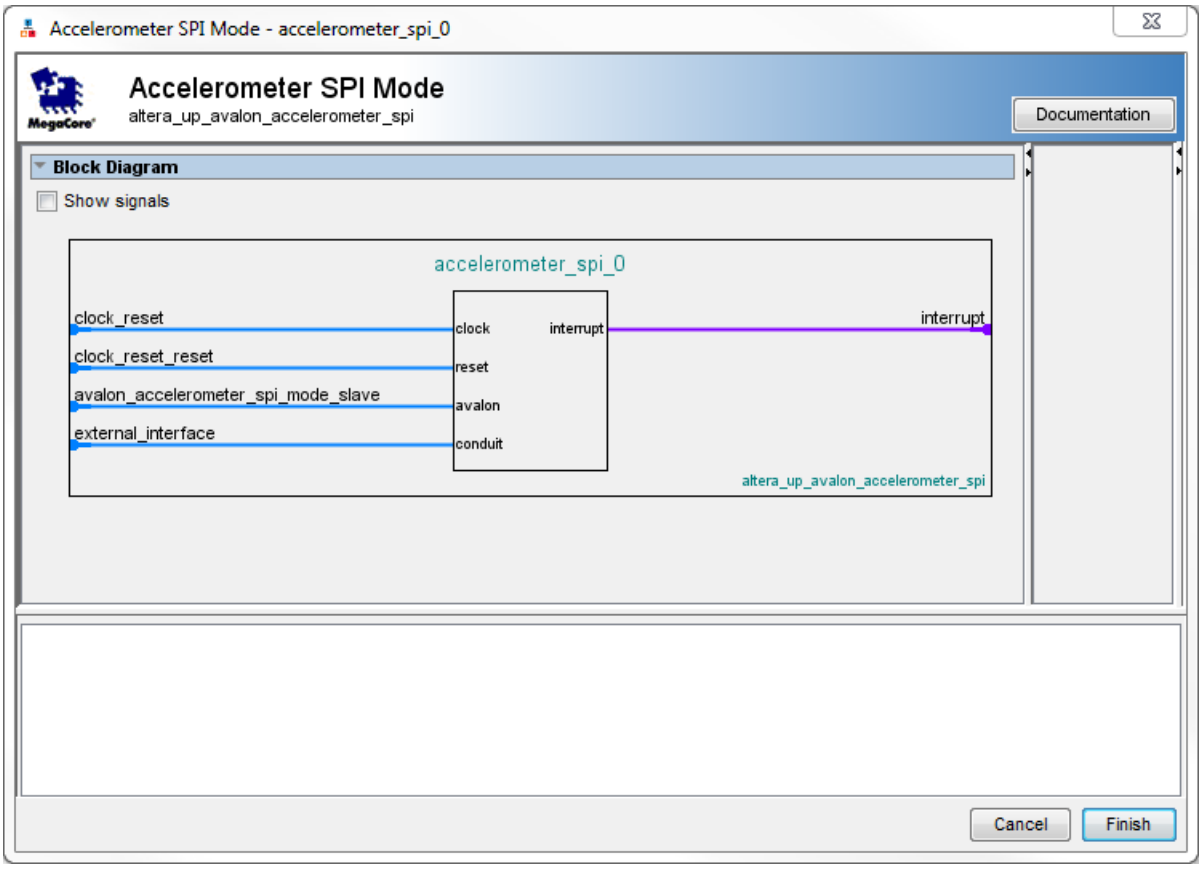

Figure 2. Configuration Wizard for the UP Accelerometer Core.

4 Software Programming Model

The UP Accelerometer Core can be controlled by a program running on a processor. This is accomplished by reading from and writing to two memory-mapped registers in the core. This can be done by directly accessing these registers. Alternatively, for the Nios II processor, those registers can be manipulated using C-language functions accessible through the Hardware Abstraction Layer (HAL) interface.

In the following sections, we describe the memory-mapped registers and how to use them, both directly and using the HAL interface.

4.1 Registers in the UP Accelerometer Core

The UP Accelerometer Core has two 8-bit registers, called *address* and *data*. They are mapped to specific memory addresses at the time the core is instantiated in a Qsys-developed system. Their addresses are offset by one byte, as indicated in Table 1. The table shows the byte offset, access rights (Read/Write) and the meaning of each bit in the registers.

By writing into these registers, the processor can specify the operationg mode of the accelerometer and read its current values.

Table 1. UP Accelerometer Core Registers

Offset in bytes	Register Name	Read/Write	7...6	5...0
0	`address`	W	0	Addr
1	`data`	R/W	Data	

4.1.1 *Address* Register

The `address` register is used to transmit the address of a register in the ADXL345 device that is to be accessed. The desired address is is given in the six least-significant bits, as shown in Table 1.

4.1.2 *Data* Register

The `data` register is used to transmit data from/to the registers in the ADXL345 device. Reading from or writing to this register will initiate a serial transfer from/to the accelerometer.

4.2 Registers in the ADXL345 Device

The ADXL345 chip has a number of control and data registers, which are identified by six-bit addresses in the range 0x00 to 0x39. All registers are eight bits long. The control registers are used to define various modes of operation. They are loaded by performing a write operation. Their contents can be examined by performing a read operation. The data registers comprise three pairs of registers, which hold the currently measured values of acceleration in the x, y and z directions.

Data registers *DATAX0* and *DATAX1*, with addresses 0x32 and 0x33, hold the 2's-complement value of the x-axis measurement. The *DATAX0* register holds the low-order byte and *DATAX1* holds the high-order byte. Similarly,

DATAY0 and *DATAY1*, with addresses 0x34 and 0x35, hold the *y*-axis value, while *DATAZ0* and *DATAZ1*, with addresses 0x36 and 0x37, hold the *z*-axis value.

4.3 Configuring the Accelerometer

The accelerometer chip has to be configured to perform the desired functions, which is accomplished by writing into the appropriate control registers. Using the UP Accelerometer Core, this can be achieved by writing the necessary addresses and data into the *address* and *data* registers in the core. However, the UP Accelerator Core has an auto-initialize option, which automatically configures it into a mode that causes measurements to be continuously performed and allows reading of the results through the X, Y and Z register pairs using the resolution of ± 2 g.

The automatic initialization is performed whenever the core is reset, which also occurs when the hardware system that contains the core is downloaded into the FPGA device on the board.

5 Device Driver for the Nios II Processor

For use with the Nios II processor, the UP Accelerometer Core is packaged with C-language functions accessible through the hardware abstraction layer (HAL). These functions enable users to communicate with the accelerometer based on the register address and data given. Users should refer to the description of registers in the ADXL345 datasheets.

To use the functions, the C code must include the statement:

```
#include "altera_up_avalon_accelerometer_spi.h"
```

5.1 alt_up_accelerometer_spi_open_dev

Prototype: `alt_up_accelerometer_spi_dev* alt_up_accelerometer_spi_open_dev(const char *name)`
Include: `<altera_up_avalon_accelerometer_spi_spi.h>`
Parameters: `name` – the accelerometer_spi component name in Qsys.
Returns: The corresponding device structure, or NULL if the device is not found.
Description: Opens the accelerometer_spi device specified by *name* .

5.2 alt_up_accelerometer_spi_read_address_register

Prototype: `int alt_up_accelerometer_spi_read_address_register(alt_up_accelerometer_sp`
`*accel, alt_u8 *addr)`
Include: `<altera_up_avalon_accelerometer_spi.h>`
Parameters: `accel` – the device structure
`addr` – a pointer to the location where the read address should be stored
Returns: 0 for success
Description: Reads configuration data from one of the accelerometer's registers.

5.3 alt_up_accelerometer_spi_read

Prototype: `int alt_up_accelerometer_spi_read(alt_up_accelerometer_spi_dev`
`*accel_spi, alt_u8 addr, alt_u8 *data)`
Include: `<altera_up_avalon_accelerometer.h>`
Parameters: `accel` – the device structure
`addr` – the accelerometer register's address
`data` – a pointer to the location where the read data should be stored
Returns: 0 for success
Description: Reads data from the accelerometer's registers.

5.4 alt_up_accelerometer_spi_write

Prototype: `int alt_up_accelerometer_spi_write(alt_up_accelerometer_spi_dev`
`*accel, alt_u8 addr, alt_u8 data)`
Include: `<altera_up_avalon_accelerometer.h>`
Parameters: `accel` – the device structure
`addr` – the accelerometer register's address
`data` – the data to be written
Returns: 0 for success
Description: Writes data to the accelerometer's registers.

5.5 alt_up_accelerometer_spi_read_x_axis

Prototype: `int alt_up_accelerometer_spi_read_x_axis(alt_up_accelerometer_spi_dev`
`*accel_spi, alt_32 *x_axis)`
Include: `<altera_up_avalon_accelerometer.h>`
Parameters: `accel` – the device structure
`x_axis` – a pointer to the location where the *x*-axis data should be stored
Returns: 0 for success
Description: Reads the *x*-xis value from both registers, *DATAX0* and *DATAX1*, and converts the value to a signed integer.

5.6 alt_up_accelerometer_spi_read_y_axis

Prototype: `int alt_up_accelerometer_spi_read_y_axis(alt_up_accelerometer_spi_dev *accel, alt_32 *y_axis)`
Include: `<altera_up_avalon_accelerometer.h>`
Parameters: `accel` – the device structure
`y_axis` – a pointer to the location where the *y*-axis data should be stored
Returns: 0 for success
Description: Reads the *y*-axis value from both registers, *DATAY0* and *DATAY1*, and converts the value to a signed integer.

5.7 alt_up_accelerometer_spi_read_z_axis

Prototype: `int alt_up_accelerometer_spi_read_z_axis(alt_up_accelerometer_spi_dev *accel, alt_32 *z_axis)`
Include: `<altera_up_avalon_accelerometer.h>`
Parameters: `accel` – the device structure
`z_axis` – a pointer to the location where the *z*-axis data should be stored
Returns: 0 for success
Description: Reads the *z*-axis value from both registers, *DATAZ0* and *DATAZ1*, and converts the value to a signed integer.