Application Note: AN00181

xCORE-200 explorer - Accelerometer

This application note show how to the accelerometer on an xCORE-200 explorer development kit. The kit itself has a Freescale FXOS8700CQ 6-Axis sensor with interated linear accelerometer and magnetometer.

The example uses the XMOS I2C library to demonstrate how I2C devices can be accessed in an easy and efficient manner. It shows how to access the registers of an I2C device connected to the GPIO of an XMOS multicore micro controller.

The code in the example builds a simple application which configures the FXOS8700CQ accelerometer and reports x,y and z acceleration values to the user. Data is output to the development console using xSCOPE and the accelerometer state is also reported via the RGB LED on the xCORE-200 explorer board.

Required tools and libraries

- xTIMEcomposer Tools - Version 14.0
- XMOS I2C library - Version 2.0.0

Required hardware

This application note is designed to run on any XMOS multicore microcontroller.

The example code provided with the application has been implemented and tested on the xCORE-200 explorer kit. The dependancy on this board is due to the FXOS8700CQ accelerometer being connected to the specific GPIO ports defined in the example. The same device could easily be added to another XMOS development platform.

Prerequisites

- This document assumes familiarity with the XMOS xCORE architecture, the XMOS GPIO library, the XMOS tool chain and the xC language. Documentation related to these aspects which are not specific to this application note are linked to in the references appendix.
- For descriptions of XMOS related terms found in this document please see the XMOS Glossary[1].
- For the information relating to the I2C library, please see the document XMOS GPIO Library[2].
- For the Freescale FXOS8700CQ device see the published datasheet[3].

[1] http://www.xmos.com/published/glossary
[2] http://www.xmos.com/published/xmos-gpio-lib
[3] http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=FXOS8700CQ

1 Overview

1.1 Introduction

xCORE-200 explorerKIT contains everything you need to start developing applications on the powerful xCORE-200 multicore microcontroller products from XMOS. It's easy to use and provides lots of advanced features on a small, low cost platform.

The xCORE-200 explorerKIT features our XE216-512 xCORE-200 multicore microcontroller. This device has sixteen 32bit logical cores that deliver up to 2000MIPS completely deterministically. The combination of 100/1000 Mbps Ethernet, high speed USB and 53 high performance GPIO make the xCORE-200 explorerKIT an ideal platform for functions ranging from robotics and motion control to networking and digital audio.

The xCORE-200 explorerKIT also features a 3D accelerometer, a 3-axis gyroscope and six servo interfaces for rapid prototyping of motor and motion control projects.

The FXOS8700CQ 6-axis sensor combines industry leading accelerometer and magnetometer sensors in a small 3 x 3 x 1.2 mm QFN plastic package. The 14-bit accelerometer and 16-bit magnetometer are combined with a high-performance ASIC to enable an eCompass solution capable of a typical orientation resolution of 0.1 degrees and sub 5 degree compass heading accuracy for most applications.

This application note demonstrates how to interface the FXOS8700CQ device with an XMOS multicore microcontroller via an I2C interface.

1.2 Block diagram

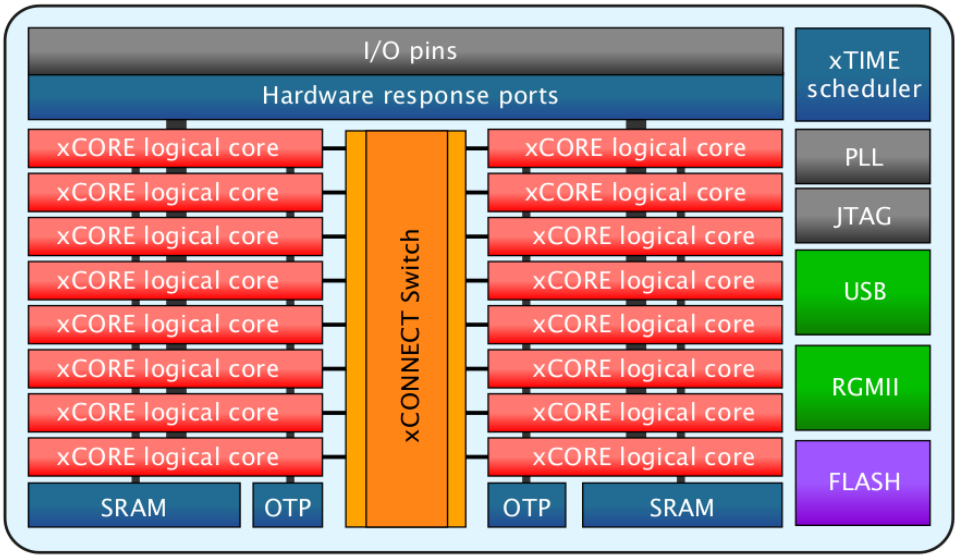

Figure 1: Block diagram of XE216-512 device on xCORE-200 explorerKIT

2 Accelerometer application note

The example in this application note uses the XMOS I2C library and shows a simple program that configures and reads accelerometer data from the attached FXOS8700CQ device.

For the accelerometer application example, the system comprises of single task running on an xCORE-200 multicore microcontroller to provide the I2C master interface and another task running to provide the application which handles the data from the accelerometer.

The tasks perform the following operations.

- A task to handle the I2C master interface
- A task to handle configuring and accessing the accelerometer in the FXOS8700CQ device

The following diagram shows the task and communication structure for this accelerometer example

Figure 2: Task diagram of the accelerometer example

2.1 Makefile additions for this example

To start using the I2C library, you need to add `lib_i2c` to your `Makefile`:

```
USED_MODULES = ... lib_i2c ...
```

You can then access the I2C functions in your source code via the `i2c.h` header file:

```
#include <i2c.h>
```

2.2 Application resource declaration

The following code declares the XMOS ports used for the I2C master interface connected to the accelerometer device.

```
// I2C interface ports
port p_scl = XS1_PORT_1E;
port p_sda = XS1_PORT_1F;
port p_led = XS1_PORT_4F;
```

These declare ports XS1_PORT_1E and XS1_PORT_1F for the I2C interface and port XS1_PORT_4F for the RBG LED interface.

There are a number of device specific defines declared in the source code, these are as follow and have been taken from the datasheet for the FXOS8700CQ device which has been referenced on the front page of the application note.

```
// FXOS8700EQ register address defines
#define FXOS8700EQ_I2C_ADDR 0x1E
#define FXOS8700EQ_XYZ_DATA_CFG_REG 0x0E
#define FXOS8700EQ_CTRL_REG_1 0x2A
#define FXOS8700EQ_DR_STATUS 0x0
#define FXOS8700EQ_OUT_X_MSB 0x1
#define FXOS8700EQ_OUT_X_LSB 0x2
#define FXOS8700EQ_OUT_Y_MSB 0x3
#define FXOS8700EQ_OUT_Y_LSB 0x4
#define FXOS8700EQ_OUT_Z_MSB 0x5
#define FXOS8700EQ_OUT_Z_LSB 0x6
```

2.3 The application main() function

Below is the source code for the main function of this application, which is taken from the source file `main.xc`

```
int main(void) {
  i2c_master_if i2c[1];
  par {
    i2c_master(i2c, 1, p_scl, p_sda, 10);
    accelerometer(i2c[0]);
  }
  return 0;
}
```

Looking at this in a more detail you can see the following:

- A i2c_master_if typed interface is declared to access the connected I2C device
- The par functionality describes running separate tasks in parallel

- The i2c_master task is combined with the accelerometer routine by the compiler
- There is a function call to configure the I2C master interface i2c_master()
- There is a function to deal with handling the accelerometer data accelerometer()
- In this example all tasks run on the tile tile[0]

2.4 Configuring the FXOS8700EQ accelerometer device

The task accelerometer() is used to handle the configuration of the device attached via the I2C master interface. There are 2 configuration registers needed to be written to enable the accelerometer and start it producing data that can be read. These can be seen in the following code.

```
// Configure FXOS8700EQ
result = i2c.write_reg(FXOS8700EQ_I2C_ADDR, FXOS8700EQ_XYZ_DATA_CFG_REG, 0x01);
if (result != I2C_REGOP_SUCCESS) {
  printf("I2C write reg failed\n");
}

// Enable FXOS8700EQ
result = i2c.write_reg(FXOS8700EQ_I2C_ADDR, FXOS8700EQ_CTRL_REG_1, 0x01);
if (result != I2C_REGOP_SUCCESS) {
  printf("I2C write reg failed\n");
}
```

The first register write configures the mode of the device and the second write enables it. The functionality that can be accessed via these registers can be found in the FXOS8700EQ datasheet.

2.5 The accelerometer data processing main loop

Once the FXOS8700EQ device is configured it is possible to read and process the data it provides for acceleration in the X,Y and Z axis. The code for the data processing main loop is as follows.

```
while (1) {
  // Wait for data ready from FXOS8700EQ
  do {
    status_data = i2c.read_reg(FXOS8700EQ_I2C_ADDR, FXOS8700EQ_DR_STATUS, result);
  } while (!status_data & 0x08);

  int x,y,z;

  x = read_acceleration(i2c, FXOS8700EQ_OUT_X_MSB);

  y = read_acceleration(i2c, FXOS8700EQ_OUT_Y_MSB);

  z = read_acceleration(i2c, FXOS8700EQ_OUT_Z_MSB);

  output_accelerometer_values(x,y,z);
}
```

From this you can see the following.

- The application checks for data ready by reading the FXOS8700EQ status register
- When the device signals that data is availble the code starts reading data values
- Data values for each of the axis is read via the read_acceleration() function
- The device register that is going to be read is passed to read_acceleration()
- Once all 3 axis have been read the data values returned are passed to output_accelerometer_values()

2.6 Reading acceleration data

The function `read_acceleration()` is used to access the FXOS8700EQ device and read data acceleration values.

The code for this function is as follow.

```
int read_acceleration(client interface i2c_master_if i2c, int reg) {
    i2c_regop_res_t result;
    int accel_val = 0;
    unsigned char data = 0;

    // Read MSB data
    data = i2c.read_reg(FXOS8700EQ_I2C_ADDR, reg, result);
    if (result != I2C_REGOP_SUCCESS) {
      printf("I2C read reg failed\n");
      return 0;
    }

    accel_val = data << 2;

    // Read LSB data
    data = i2c.read_reg(FXOS8700EQ_I2C_ADDR, reg+1, result);
    if (result != I2C_REGOP_SUCCESS) {
      printf("I2C read reg failed\n");
      return 0;
    }

    accel_val |= (data >> 6);

    if (accel_val & 0x200) {
      accel_val -= 1023;
    }

    return accel_val;
}
```

You can see the following in the code

- The axis to be read is passed into the function in the `reg` parameter
- There is an I2C read from the MSB register into the variable `data`
- The data value is processed into the return value `accel_val`
- There is an I2C read from the LSB register into the variable `data`
- The LSB data value is combined with the MSB data value in the variable `accel_val`
- The function processes and returns the value of the requested axis

2.7 Outputting accelerometer data

The data processing main loop calls the function `output_accelerometer_values()` to send data to the development console via xSCOPE and to display state on the RGB LED.

The code for this function is as follows.

```
void output_accelerometer_values(int x, int y, int z) {
  int rgb_led_value = 0;

  if (x > 0) {
    rgb_led_value |= 0x2;
  }
  if (y > 0) {
    rgb_led_value |= 0x4;
  }
  if (z > 0) {
    rgb_led_value |= 0x8;
  }

  p_led <: rgb_led_value;

  printf("X = %d, Y = %d, Z = %d       \r", x, y, z);
}
```

To output the accelerometer values the code does the following.

- If the x axis value is positive it sets the red led on, otherwise it turns it off.
- If the y axis value is positive it sets the green led on, otherwise it turns it off.
- If the z axis value is positive it sets the blue led on, otherwise it turns it off.
- The value of the RGDB led is output to the port p_led
- `printf()` is used to send the x,y and z values to the development console.

APPENDIX A - Demo Hardware Setup

To run the demo, connect the xCORE-200 explorerKIT power to a USB socket, plug the XTAG into the board and connect the xTAG USB cable to your development machine

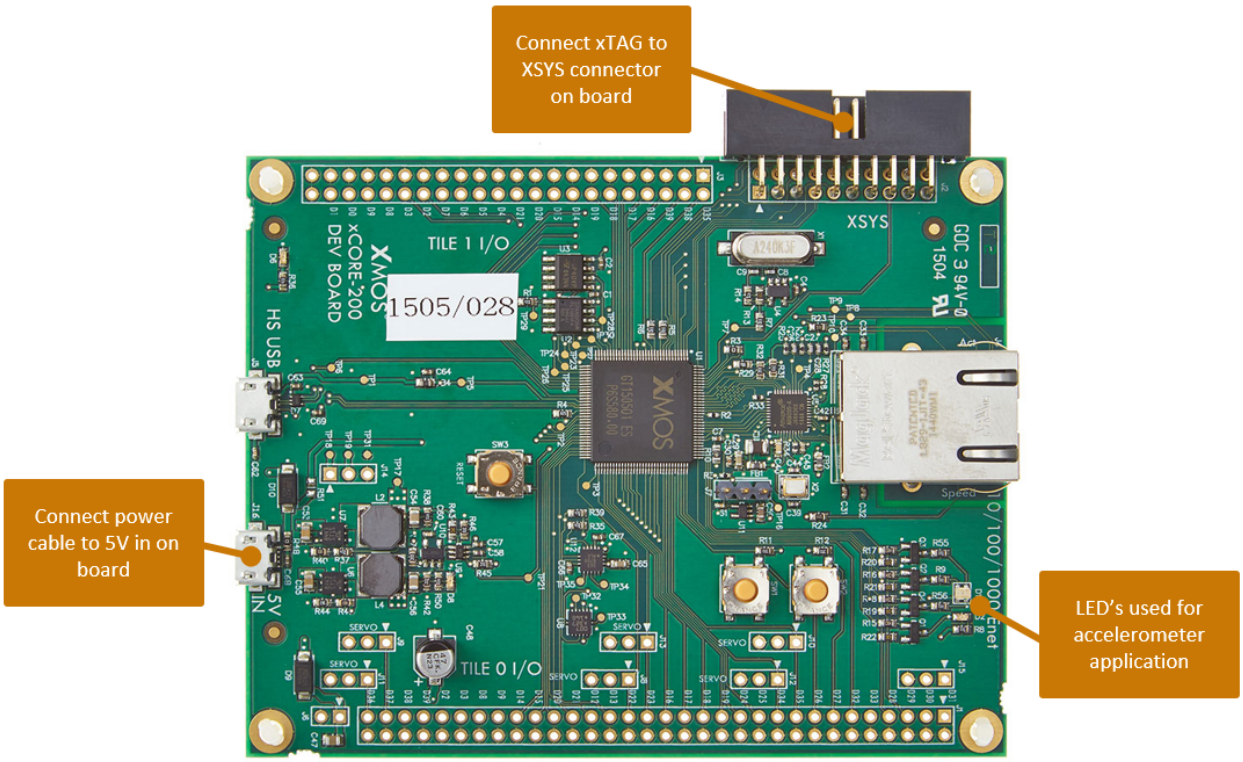

Figure 3: XMOS xCORE-200 explorerKIT

The hardware should be configured as displayed above for this demo:

- The XTAG debug adapter should be connected to the XSYS connector and the XTAG USB cable should be connected to the host machine
- The xCORE-200 explorerKIT should have the power cable connected

APPENDIX B - Launching the demo application

Once the demo example has been built either from the command line using xmake or via the build mechanism of xTIMEcomposer studio we can execute the application on the xCORE-USB sliceKIT.

Once built there will be a `bin` directory within the project which contains the binary for the xCORE device. The xCORE binary has a XMOS standard .xe extension.

B.1 Launching from the command line

From the command line we use the `xrun` tool to download code to both the xCORE devices. If we change into the bin directory of the project we can execute the code on the xCORE microcontroller as follows:

```
> xrun --xscope app_acccelerometer_demo.xe          <-- Download and execute the xCORE code
```

Once this command has executed the application will be running on the xCORE-200 explorerKIT

B.2 Launching from xTIMEcomposer Studio

From xTIMEcomposer Studio we use the run mechanism to download code to xCORE device. Select the xCORE binary from the bin directory, right click and go to `Run Configurations`. Double click on xCORE application to create a new run configuration, enable the xSCOPE I/O mode in the dialog box and then select Run.

Once this command has executed the application will be running on the xCORE-200 explorerKIT

B.3 Running the accelerometer demo

Once the application is started via either of the above methods there should be output printed to the console showing the x,y and z axis values and as you move the development board these will change along with the RGB LED on the development board.

APPENDIX C - References

XMOS Tools User Guide

http://www.xmos.com/published/xtimecomposer-user-guide

XMOS xCORE Programming Guide

http://www.xmos.com/published/xmos-programming-guide

XMOS I2C Library

http://www.xmos.com/published/xmos-i2c-lib

Freescale FXOS8700CQ device information and datasheet

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=FXOS8700CQ

APPENDIX D - Full source code listing

D.1 Source code for main.xc

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved

#include <xs1.h>
#include <stdio.h>
#include "i2c.h"
#include <xscope.h>

// I2C interface ports
port p_scl = XS1_PORT_1E;
port p_sda = XS1_PORT_1F;
port p_led = XS1_PORT_4F;

// FXOS8700EQ register address defines
#define FXOS8700EQ_I2C_ADDR 0x1E
#define FXOS8700EQ_XYZ_DATA_CFG_REG 0x0E
#define FXOS8700EQ_CTRL_REG_1 0x2A
#define FXOS8700EQ_DR_STATUS 0x0
#define FXOS8700EQ_OUT_X_MSB 0x1
#define FXOS8700EQ_OUT_X_LSB 0x2
#define FXOS8700EQ_OUT_Y_MSB 0x3
#define FXOS8700EQ_OUT_Y_LSB 0x4
#define FXOS8700EQ_OUT_Z_MSB 0x5
#define FXOS8700EQ_OUT_Z_LSB 0x6

int read_acceleration(client interface i2c_master_if i2c, int reg) {
    i2c_regop_res_t result;
    int accel_val = 0;
    unsigned char data = 0;

    // Read MSB data
    data = i2c.read_reg(FXOS8700EQ_I2C_ADDR, reg, result);
    if (result != I2C_REGOP_SUCCESS) {
      printf("I2C read reg failed\n");
      return 0;
    }

    accel_val = data << 2;

    // Read LSB data
    data = i2c.read_reg(FXOS8700EQ_I2C_ADDR, reg+1, result);
    if (result != I2C_REGOP_SUCCESS) {
      printf("I2C read reg failed\n");
      return 0;
    }

    accel_val |= (data >> 6);

    if (accel_val & 0x200) {
      accel_val -= 1023;
    }

    return accel_val;
}

void output_accelerometer_values(int x, int y, int z) {
  int rgb_led_value = 0;

  if (x > 0) {
    rgb_led_value |= 0x2;
  }
  if (y > 0) {
    rgb_led_value |= 0x4;
  }
  if (z > 0) {
    rgb_led_value |= 0x8;
  }

  p_led <: rgb_led_value;
```

```
  printf("X = %d, Y = %d, Z = %d       \r", x, y, z);
}

void accelerometer(client interface i2c_master_if i2c) {
  i2c_regop_res_t result;
  char status_data = 0;

  // Configure FXOS8700EQ
  result = i2c.write_reg(FXOS8700EQ_I2C_ADDR, FXOS8700EQ_XYZ_DATA_CFG_REG, 0x01);
  if (result != I2C_REGOP_SUCCESS) {
    printf("I2C write reg failed\n");
  }

  // Enable FXOS8700EQ
  result = i2c.write_reg(FXOS8700EQ_I2C_ADDR, FXOS8700EQ_CTRL_REG_1, 0x01);
  if (result != I2C_REGOP_SUCCESS) {
    printf("I2C write reg failed\n");
  }

  while (1) {
    // Wait for data ready from FXOS8700EQ
    do {
      status_data = i2c.read_reg(FXOS8700EQ_I2C_ADDR, FXOS8700EQ_DR_STATUS, result);
    } while (!status_data & 0x08);

    int x,y,z;

    x = read_acceleration(i2c, FXOS8700EQ_OUT_X_MSB);

    y = read_acceleration(i2c, FXOS8700EQ_OUT_Y_MSB);

    z = read_acceleration(i2c, FXOS8700EQ_OUT_Z_MSB);

    output_accelerometer_values(x,y,z);
  }
  // End of accelerometer()
}

int main(void) {
  i2c_master_if i2c[1];
  par {
    i2c_master(i2c, 1, p_scl, p_sda, 10);
    accelerometer(i2c[0]);
  }
  return 0;
}
```