Application Note: AN00156

How to use the I2C master library

Required tools and libraries

The code in this application note is known to work on version 14.2.3 of the xTIMEcomposer tools suite, it may work on other versions.

The application depends on the following libraries:

- lib_i2c (>=4.0.0)
- lib_logging (>=2.1.0)

Required hardware

The example code provided with the application has been implemented and tested on the xCORE-200 eXplorerKIT.

Prerequisites

- This document assumes familiarity with the XMOS xCORE architecture, 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 a description of XMOS related terms found in this document please see the XMOS Glossary[1].

[1] http://www.xmos.com/published/glossary

1 Overview

1.1 Introduction

The XMOS I^{2}C library provides software defined, industry-standard, I^{2}C components that allow you to create devices which can be either I^{2}C bus masters or slaves using xCORE GPIO ports.

I^{2}C is a two-wire bus with defined protocols for connecting devices. There is a clock and a data line, both of which are pulled high by external pull-up resistors and driven low by the I^{2}C devices.

The XMOS I^{2}C library includes support for master and slave devices at speeds of up to 400kb/s.

This application note demonstrates using the I^{2}C master library to work with the FXOS8700CQ accelerometer device on the xCORE-200 eXplorerKIT board.

1.2 Block diagram

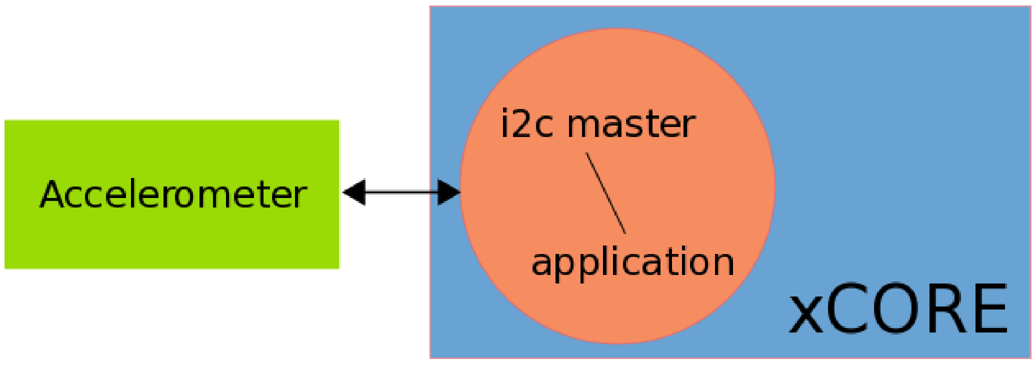

Figure 1: Application block diagram

The application uses a single logical core which runs the application and makes calls to the I^{2}C master library functions as required.

2 How to use I^{2}C master

2.1 The Makefile

To start using the I^{2}C, you need to add lib_i2c to you Makefile:

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

This demo also uses the logging library (lib_logging) for the debug_printf function. This is a faster, but more limited version of the C-Standard Library printf function. So the Makefile also includes:

```
USED_MODULES = .. lib_logging ..
```

The logging library is configurable at compile-time allowing calls to debug_printf() to be easily enabled or disabled. For the prints to be enabled it is necessary to add the following to the compile flags:

```
XCC_FLAGS = .. -DDEBUG_PRINT_ENABLE=1 ..
```

2.2 Includes

This application requires the system header that defines XMOS xCORE specific defines for declaring and initialising hardware:

```
#include <xs1.h>
```

The I^{2}C library functions are defined in i2c.h. This header must be included in your code to use the library. The logging functions are provided by debug_print.h.

```
#include "i2c.h"
#include "debug_print.h"
```

2.3 Allocating hardware resources

An I^{2}C master requires a clock and a data pin. On an xCORE the pins are controlled by ports. The application therefore declares two 1-bit ports:

```
port p_scl = XS1_PORT_1E;
port p_sda = XS1_PORT_1F;
```

2.4 Accelerometer defines

A number of defines are used for the accelerometer device address and register numbers:

```
```

2.5 Reading over I^{2}C

The read_acceleration() function is used to get an accelerometer reading for a given axis. It uses the I^{2}C master to read the MSB and LSB registers and then combines their results into the 10-bit value for the specified axis. Each register read is checked to ensure that it has completed correctly.

```
int read_acceleration(client interface i2c_master_if i2c, int axis) {
    i2c_regop_res_t result;

    uint8_t msb_data = i2c.read_reg(FXOS8700CQ_I2C_ADDR, axis, result);
    if (result != I2C_REGOP_SUCCESS) {
      debug_printf("I2C read reg failed\n");
      return 0;
    }

    uint8_t lsb_data = i2c.read_reg(FXOS8700CQ_I2C_ADDR, axis+1, result);
    if (result != I2C_REGOP_SUCCESS) {
      debug_printf("I2C read reg failed\n");
      return 0;
    }

    int accel_val = (msb_data << 2) | (lsb_data >> 6);
    if (accel_val & 0x200) {
      accel_val -= 1023;
    }
    return accel_val;
}
```

The I^{2}C read_reg() function takes the device address, the register number to read and a variable in which to return whether the read was successful.

By default it is assumed that the device address, register number and data are all 8-bit. The I^{2}C library provides other functions with different data-width operands. Refer to the library documentation for details.

2.6 Writing over I^{2}C

The core of the application is the accelerometer() function which starts by writing to the accelerometer device to configure and then enable it:

```
void accelerometer(client interface i2c_master_if i2c) {
  i2c_regop_res_t result;

  // Configure FXOS8700CQ
  result = i2c.write_reg(FXOS8700CQ_I2C_ADDR, FXOS8700CQ_XYZ_DATA_CFG_REG, 0x01);
  if (result != I2C_REGOP_SUCCESS) {
    debug_printf("I2C write reg failed\n");
  }

  // Enable FXOS8700CQ
  result = i2c.write_reg(FXOS8700CQ_I2C_ADDR, FXOS8700CQ_CTRL_REG_1, 0x01);
  if (result != I2C_REGOP_SUCCESS) {
    debug_printf("I2C write reg failed\n");
  }
```

After that it continually loops polling the accelerometer until it is ready and then reading the values from the three axes and displaying the current status.

```
while (1) {
  // Wait for data ready from FXOS8700CQ
  char status_data = 0;
  do {
    status_data = i2c.read_reg(FXOS8700CQ_I2C_ADDR, FXOS8700CQ_DR_STATUS, result);
  } while (!status_data & 0x08);

  int x = read_acceleration(i2c, FXOS8700CQ_OUT_X_MSB);
  int y = read_acceleration(i2c, FXOS8700CQ_OUT_Y_MSB);
  int z = read_acceleration(i2c, FXOS8700CQ_OUT_Z_MSB);

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

The print uses a \r to ensure that only a single line of the screen is used.

2.7 The application main() function

The main() function sets up the tasks in the application.

Firstly, the interface is declared. In xC interfaces provide a means of concurrent tasks communicating with each other. In this application there is a single interface between the application and the I^{2}C master.

```
i2c_master_if i2c[1];
```

The rest of the main() function starts all the tasks in parallel using the xC par construct:

```
par {
  i2c_master(i2c, 1, p_scl, p_sda, 10);
  accelerometer(i2c[0]);
}
```

This code starts the I^{2}C master and the application. Because the I^{2}C master is marked as distributable it will not actually use a logical core but will be run on the logical cores with the application. As a result, the entire system only requires on logical core.

APPENDIX A - Demo Hardware Setup

To run the demo, connect a USB cable to power the xCORE-200 eXplorerKIT and plug the xTAG to the board and connect the xTAG USB cable to your development machine.

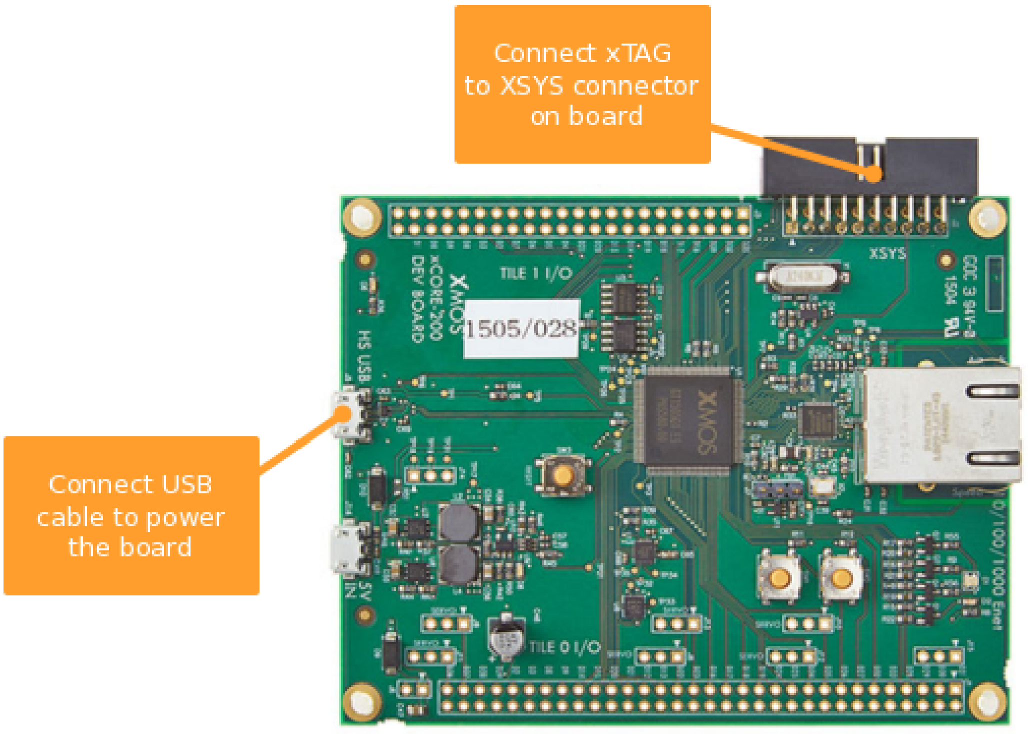
Connect xTAG to XSYS connector on board
Connect USB cable to power the board

Figure 2: Hardware setup

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 it can be executed on the xCORE-200 eXplorerKIT.

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 you use the xrun tool to download and run the code on the xCORE device:

```
xrun --xscope bin/AN00156_i2c_master_example.xe
```

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

B.2 Launching from xTIMEcomposer Studio

From xTIMEcomposer Studio 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 application

Once the application is started using 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.

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 I^{2}C Library

http://www.xmos.com/support/libraries/lib_i2c

APPENDIX D - Full source code listing

D.1 Source code for main.xc

```
// Copyright (c) 2013-2016, XMOS Ltd, All rights reserved
#include <xs1.h>
#include "i2c.h"
#include "debug_print.h"

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

// FXOS8700CQ register address defines
#define FXOS8700CQ_I2C_ADDR 0x1E
#define FXOS8700CQ_XYZ_DATA_CFG_REG 0x0E
#define FXOS8700CQ_CTRL_REG_1 0x2A
#define FXOS8700CQ_DR_STATUS 0x0
#define FXOS8700CQ_OUT_X_MSB 0x1
#define FXOS8700CQ_OUT_Y_MSB 0x3
#define FXOS8700CQ_OUT_Z_MSB 0x5

int read_acceleration(client interface i2c_master_if i2c, int axis) {
    i2c_regop_res_t result;

    uint8_t msb_data = i2c.read_reg(FXOS8700CQ_I2C_ADDR, axis, result);
    if (result != I2C_REGOP_SUCCESS) {
      debug_printf("I2C read reg failed\n");
      return 0;
    }

    uint8_t lsb_data = i2c.read_reg(FXOS8700CQ_I2C_ADDR, axis+1, result);
    if (result != I2C_REGOP_SUCCESS) {
      debug_printf("I2C read reg failed\n");
      return 0;
    }

    int accel_val = (msb_data << 2) | (lsb_data >> 6);
    if (accel_val & 0x200) {
      accel_val -= 1023;
    }
    return accel_val;
}

void accelerometer(client interface i2c_master_if i2c) {
  i2c_regop_res_t result;

  // Configure FXOS8700CQ
  result = i2c.write_reg(FXOS8700CQ_I2C_ADDR, FXOS8700CQ_XYZ_DATA_CFG_REG, 0x01);
  if (result != I2C_REGOP_SUCCESS) {
    debug_printf("I2C write reg failed\n");
  }

  // Enable FXOS8700CQ
  result = i2c.write_reg(FXOS8700CQ_I2C_ADDR, FXOS8700CQ_CTRL_REG_1, 0x01);
  if (result != I2C_REGOP_SUCCESS) {
    debug_printf("I2C write reg failed\n");
  }

  while (1) {
    // Wait for data ready from FXOS8700CQ
    char status_data = 0;
    do {
      status_data = i2c.read_reg(FXOS8700CQ_I2C_ADDR, FXOS8700CQ_DR_STATUS, result);
    } while (!status_data & 0x08);

    int x = read_acceleration(i2c, FXOS8700CQ_OUT_X_MSB);
    int y = read_acceleration(i2c, FXOS8700CQ_OUT_Y_MSB);
    int z = read_acceleration(i2c, FXOS8700CQ_OUT_Z_MSB);

    debug_printf("X = %d, Y = %d, Z = %d       \r", x, y, z);
  }
  // End accelerometer
}

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