Application Note: AN00157

How to use the I2C slave library

Required tools and libraries

The code in this application note is known to work on version 14.1.1 of the xTIMEcomposer tools suite, it may work on other versions.

The application depends on the following libraries:

- lib_logging
- lib_i2c

Required hardware

Prerequisites

- This document assumes familiarity with the XMOS xCORE architecture, the Universal Serial Bus 2.0 Specification (and related specifications, 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 Overview

1.1 Introduction

1.2 Block diagram

2 How to use I2C slave

APPENDIX A - Demo Hardware Setup

APPENDIX B - Launching the demo application

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 xCORE-USB Device Library:

http://www.xmos.com/published/xuddg

XMOS USB Device Design Guide:

http://www.xmos.com/published/xmos-usb-device-design-guide

USB HID Class Specification, USB.org:

http://www.usb.org/developers/devclass_docs/HID1_11.pdf

USB 2.0 Specification

http://www.usb.org/developers/docs/usb20_docs/usb_20_081114.zip

APPENDIX D - Full source code listing

D.1 Source code for endpoint0.xc

D.2 Source code for main.xc

```
// Copyright (c) 2015, XMOS Ltd, All rights reserved
#include <i2c.h>
#include <debug_print.h>
#include <xs1.h>
#include <syscall.h>

port p_scl = XS1_PORT_1A;
port p_sda = XS1_PORT_1B;

interface register_if {
  void set_register(int regnum, uint8_t data);
  uint8_t get_register(int regnum);
};

#define NUM_REGISTERS 10

[[distributable]]
void i2c_slave_register_file(server i2c_slave_callback_if i2c,
                             server interface register_if app)
{
  uint8_t registers[NUM_REGISTERS];

  // This variable is set to -1 if no current register has been selected.
  // If the I2C master does a write transaction to select the register then
  // the variable will be updated to the register the master wants to
  // read/update.
  int current_regnum = -1;
  while (1) {
    select {

    // Handle application requests to get/set register values.
    case app.set_register(int regnum, uint8_t data):
      registers[regnum] = data;
      break;
    case app.get_register(int regnum) -> uint8_t data:
      data = registers[regnum];
      break;

    // Handle I2C slave transactions
    case i2c.start_read_request(void):
      break;
    case i2c.ack_read_request(void) -> i2c_slave_ack_t response:
      // If the no register has been asked for via a previous write
      // transaction the NACK, otherwise ACK.
      if (current_regnum == -1) {
        response = I2C_SLAVE_NACK;
      } else {
        response = I2C_SLAVE_ACK;
      }
      break;
    case i2c.start_write_request(void):
```

```
      break;
    case i2c.ack_write_request(void) -> i2c_slave_ack_t response:
      // Write requests are always accepted.
      response = I2C_SLAVE_ACK;
      break;
    case i2c.start_master_write(void):
      break;
    case i2c.master_sent_data(uint8_t data) -> i2c_slave_ack_t response:
      // The master is trying to write, which will either select a register
      // or write to a previously selected register.
      if (current_regnum != -1) {
        registers[current_regnum] = data;
        current_regnum = -1;
        response = I2C_SLAVE_ACK;
      }
      else {
        if (data < NUM_REGISTERS)
          current_regnum = data;
        response = I2C_SLAVE_NACK;
      }
      break;
    case i2c.start_master_read(void):
      break;
    case i2c.master_requires_data() -> uint8_t data:
      // The master is trying to read, if a register is selected then
      // return the value (other return 0).
      if (current_regnum != -1) {
        data = registers[current_regnum];
      } else {
        data = 0;
      }
      break;
    case i2c.stop_bit():
      break;
    }
  }
}

void my_application(client interface register_if reg)
{
  reg.set_register(0, 0x99);
  reg.set_register(1, 0x33);
  while (1) {
    delay_milliseconds(500);
    debug_printf("Register 2 value: 0x%x\n", reg.get_register(2));
  }
}

int main() {
  i2c_slave_callback_if i_i2c;
  interface register_if i_reg;
  par {
    my_application(i_reg);
    i2c_slave_register_file(i_i2c, i_reg);
    i2c_slave(i_i2c, p_scl, p_sda, 0x3c);
  }
  return 0;
}
```