Application Note: AN00155

xCORE-200 explorer - Simple GPIO

This application note show how to use simple GPIO on an xCORE-200 explorer development kit. The kit itself contains buttons and LED's which can be accessed from application code running on the xCORE multicore microcontroller.

The example uses the XMOS GPIO library to demonstrate how simple GPIO devices can be accessed from multibit ports in an easy and efficient manner. It also demonstrates how to respond to events from within application code.

The code in the example builds a simple GPIO handling application which responds to button presses from the user and toggles the state of LED's on the development board.

Required tools and libraries

- xTIMEcomposer Tools - Version 14.0
- XMOS GPIO library - Version 1.0.0

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 only due to the GPIO ports that are connected to the buttons and LED's. These port definitions are in the source code and can be easily modified to work on another XMOS development board.

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 GPIO library, please see the document XMOS GPIO Library[2].

[1] http://www.xmos.com/published/glossary
[2] http://www.xmos.com/published/xmos-gpio-lib

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.

This application note demonstrates how to use simple GPIO and event handling with the buttons and LED's on the development kit.

1.2 Block diagram

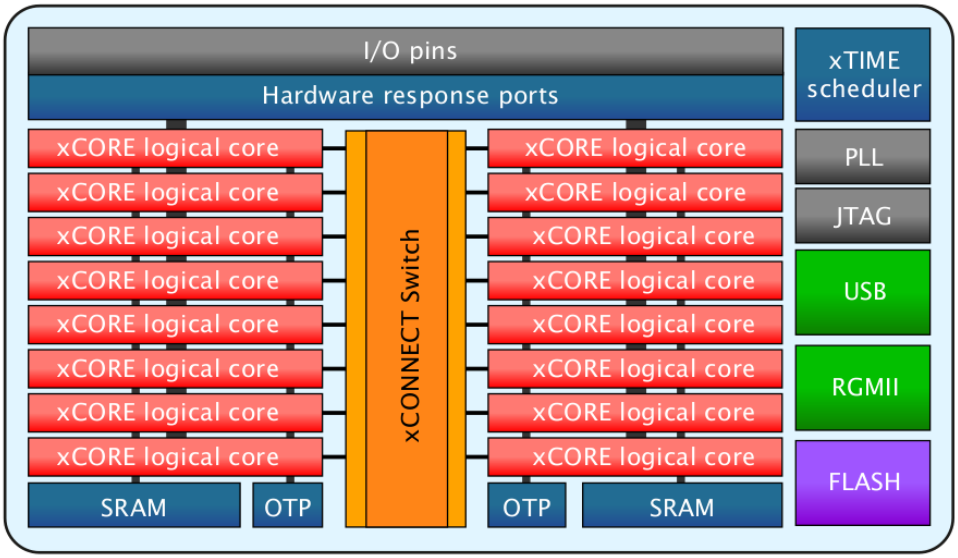

Figure 1: Block diagram of XE216-512 device on xCORE-200 explorerKIT

2 Simple GPIO application note

The example in this application note uses the XMOS GPIO library and shows a simple program that responds to user interaction on the buttons of the development board and toggles LED values based on the input.

For the Simple GPIO application example, the system comprises of two tasks running on seperate logical cores of an xCORE-200 multicore microcontroller.

The tasks perform the following operations.

- A task to handle input from a multibit port provided by the XMOS GPIO library
- A task to provide the GPIO handler routine which sets LED's on and off based on user input on the buttons

These tasks communicate via the use of interfaces which allow data to be passed between application code running on separate logical cores and tiles.

The following diagram shows the task and communication structure for this simple GPIO application example.

Figure 2: Task diagram of simple GPIO example

2.1 Makefile additions for this example

To start using the GPIO library, you need to add `lib_gpio` to your `Makefile`:

```
USED_MODULES = ... lib_gpio ...
```

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

```
#include <gpio.h>
```

2.2 Application resource declaration

The following declarations are used to define the GPIO ports used in this application example.

```
// GPIO port declarations
on tile[0] : in port explorer_buttons = XS1_PORT_4E;
on tile[0] : out port explorer_leds = XS1_PORT_4F;
```

These configure GPIO port XS1_PORT_4E to be an input port which is connect to the buttons on the physical hardware and XS1_PORT_4F as an output port which is connected to the LED's on the development board.

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() {
  input_gpio_if i_explorer_buttons[2];
  output_gpio_if i_explorer_leds[4];

  par {
     on tile[0] : input_gpio_with_events(i_explorer_buttons, 2, explorer_buttons, null);
     on tile[0] : output_gpio(i_explorer_leds, 4, explorer_leds, null);
     on tile[0] : gpio_handler(i_explorer_buttons[0], i_explorer_buttons[1],
                               i_explorer_leds[0], i_explorer_leds[1],
                               i_explorer_leds[2], i_explorer_leds[3]);
  }

   return 0;
}
```

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

- A GPIO input typed interface is declared for the two development board buttons
- A GPIO output typed interface is declared for the four development board LED's
- The par functionality describes running separate tasks in parallel
- The `output_gpio` task is combined with the `gpio_handler` routine by the compiler
- There is a function call to configure the input GPIO interfaces `input_gpio_with_events()`
- There is a function call to configure the output GPIO interfaces `output_gpio()`
- There is a function to deal with handling the button requests from the user `gpio_handler()`
- In this example all tasks run on the `tile[0]`

2.4 Responding to user interaction with the buttons

The application code for dealing with user button presses is implemented in the file `main.xc`, this contains the function `gpio_handler()` which is used to deal with events on the GPIO ports connected to the buttons.

The task defined by the function gpio_handler() takes the following arguments.

```
void gpio_handler(client input_gpio_if button_1, client input_gpio_if button_2,
                  client output_gpio_if led_green, client output_gpio_if rgb_led_blue,
                  client output_gpio_if rgb_led_green, client output_gpio_if rgb_led_red) {
```

From this you can see the following.

- The first two arguments are the input GPIO interfaces for the two buttons on the xCORE-200 explorer board
- The next four arguments are the output GPIO interfaces for the green LED and the three elements of the RGB LED

Inside this task the current state of each LED is kept in the following variables which are declared at the top of the function

```
// LED state
unsigned int green_led_state = 0;
unsigned int rgb_led_state = 0;
```

The input GPIO's are connected to the buttons on the development board, the buttons themselves are active low. The following code sets the initial event state of both of the buttons to trigger when the value on the GPIO pin is 0. This is done using the function event_when_pins_eq.

```
// Initial button event state, active low
button_1.event_when_pins_eq(0);
button_2.event_when_pins_eq(0);
```

The body of the GPIO handling task is enclosed within a while (1) loop and a select statement which triggers when events on the buttons occur.

```
while (1) {
  select {
```

There are two events handled in the select statement, the first is for button 1 on the development board. This button will be used to toggle the state of the single green LED.

```
// Triggered by events on button 1
case button_1.event():
  if (button_1.input() == 0) {
    green_led_state = ~green_led_state;
    led_green.output(green_led_state);
    // Set button event state to active high for debounce
    button_1.event_when_pins_eq(1);
  } else {
    // Debounce button
    delay_milliseconds(50);
    button_1.event_when_pins_eq(0);
  }
  break;
```

From this you can see the following.

- The event is triggered whenever the GPIO event state set with event_when_pins_eq occurs
- If the value is 0 (button pressed) the state of the green led is toggled
- The new value to be driven to the green LED is output with led_green.output(green_led_state)
- The state of the green LED is kept in the variable green_led_state
- The event on button_1 is then set to trigger when the GPIO goes high using event_when_pins_eq

- When the event triggers with a value of 1 on the GPIO we debounce the button and go back to original state

The second event handled in the `select` statement deals with button 2 on the development board. This button is used to toggle the state of the RGB LED on the development board.

```
  // Triggered by events on button 2
  case button_2.event():
    if (button_2.input() == 0) {
      rgb_led_red.output(0);
      rgb_led_green.output(0);
      rgb_led_blue.output(0);
      rgb_led_state++;
      rgb_led_state %= 4;

      switch (rgb_led_state) {
        case 1:
          rgb_led_red.output(1);
          break;
        case 2:
          rgb_led_green.output(1);
          break;
        case 3:
          rgb_led_blue.output(1);
          break;
      }
      // Set button event state to active high for debounce
      button_2.event_when_pins_eq(1);
    } else {
      // Debounce button
      delay_milliseconds(50);
      button_2.event_when_pins_eq(0);
    }
    break;
}
```

From this you can see the following.

- The event is triggered whenever the GPIO event state set with `event_when_pins_eq` occurs
- If the value is 0 (button pressed) the state of the of the RGB LED is changed
- The RGB LED state goes from RED -> GREEN -> BLUE -> OFF
- There is a basic state machine which keeps track of the RGB LED state using a variable `rgb_led_state`
- The new value to be driven to the element of the RGB LED is done via the `output()` interface function
- The event on `button_2` is then set to trigger when the GPIO goes high using `event_when_pins_eq`
- When the event triggers with a value of 1 on the GPIO we debounce the button and go back to original state

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

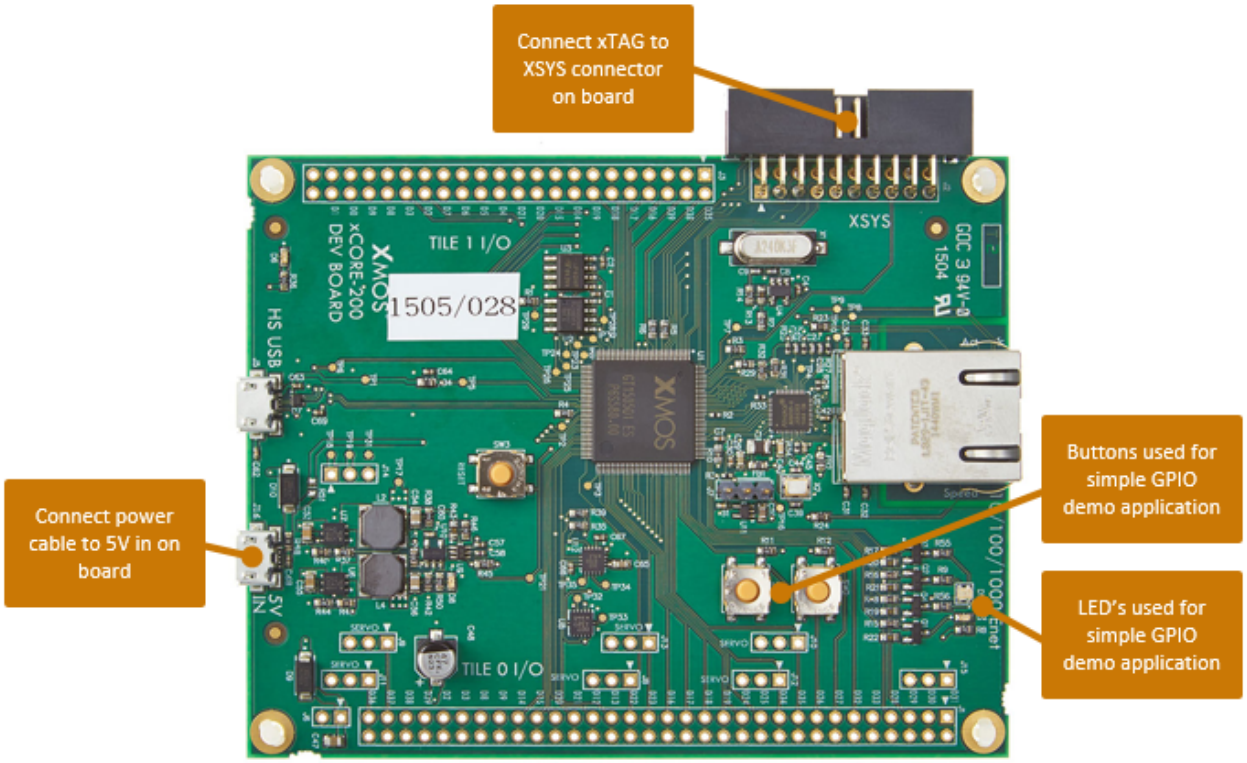

Figure 3: XMOS xCORE-200 explorerKIT

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 app_simple_gpio_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 then run as xCORE application will execute the code on the xCORE device.

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

B.3 Running the simple GPIO demo

Once the application is started via either of the above methods the LED's on the development board can be toggled by pressing the buttons. This demonstrates a simple event handling application on the xCORE-200 explorerKIT.

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 GPIO Library

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

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 <platform.h>
#include <gpio.h>

// GPIO port declarations
on tile[0] : in port explorer_buttons = XS1_PORT_4E;
on tile[0] : out port explorer_leds = XS1_PORT_4F;

// GPIO handler routine
void gpio_handler(client input_gpio_if button_1, client input_gpio_if button_2,
                  client output_gpio_if led_green, client output_gpio_if rgb_led_blue,
                  client output_gpio_if rgb_led_green, client output_gpio_if rgb_led_red) {

  // LED state
  unsigned int green_led_state = 0;
  unsigned int rgb_led_state = 0;

  // Initial button event state, active low
  button_1.event_when_pins_eq(0);
  button_2.event_when_pins_eq(0);

  while (1) {
    select {
      // Triggered by events on button 1
      case button_1.event():
        if (button_1.input() == 0) {
          green_led_state = ~green_led_state;
          led_green.output(green_led_state);
          // Set button event state to active high for debounce
          button_1.event_when_pins_eq(1);
        } else {
          // Debounce button
          delay_milliseconds(50);
          button_1.event_when_pins_eq(0);
        }
        break;

      // Triggered by events on button 2
      case button_2.event():
        if (button_2.input() == 0) {
          rgb_led_red.output(0);
          rgb_led_green.output(0);
          rgb_led_blue.output(0);
          rgb_led_state++;
          rgb_led_state %= 4;

          switch (rgb_led_state) {
            case 1:
              rgb_led_red.output(1);
              break;
            case 2:
              rgb_led_green.output(1);
              break;
            case 3:
              rgb_led_blue.output(1);
              break;
          }
          // Set button event state to active high for debounce
          button_2.event_when_pins_eq(1);
        } else {
          // Debounce button
          delay_milliseconds(50);
          button_2.event_when_pins_eq(0);
        }
        break;
    }
    // end of event select
```

```
  }

}

// The main() function runs a single task which takes the gpio interfaces as parameters
int main() {
  input_gpio_if i_explorer_buttons[2];
  output_gpio_if i_explorer_leds[4];

  par {
     on tile[0] : input_gpio_with_events(i_explorer_buttons, 2, explorer_buttons, null);
     on tile[0] : output_gpio(i_explorer_leds, 4, explorer_leds, null);
     on tile[0] : gpio_handler(i_explorer_buttons[0], i_explorer_buttons[1],
                               i_explorer_leds[0], i_explorer_leds[1],
                               i_explorer_leds[2], i_explorer_leds[3]);
  }

   return 0;
}
```