Application Note: AN00174

A startKIT glowing LED demo

This application demonstrates I/O on the startKIT by showing a glowing LED pattern on the LEDs. It uses the startKIT support library to access the I/O on the device.

Required tools and libraries

- xTIMEcomposer Tools - Version 14.0
- startKIT support library (lib_startkit_support) - Version 1.0.0

Required hardware

This application note is designed to run on the XMOS startKIT.

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].

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

1 Overview

1.1 Introduction

startKIT is a low-cost development board for the configurable xCORE multicore microcontroller products from XMOS. It's easy to use and provides lots of advanced features on a small, extremely low cost platform.

xCORE lets you software-configure the interfaces that you need for your system; so with startKIT you can configure the board to your match your exact requirements. Its 500MIPS xCORE multicore microcontroller has eight 32bit logical cores that perform deterministically, making startKIT an ideal platform for functions ranging from robotics and motion control to networking and digital audio.

startKIT also connects easily to your Raspberry Pi, allowing you to add real-time I/O and communication features to this popular computing platform, and to try out advanced applications for xCORE.

2 Glowing LEDs

The example in this application note shows off using the I/O on the startKIT and the use of an XMOS library (in particular the startKIT support library).

The example consists of a single application task which connects to a GPIO driver task supplied by the startKIT support library.

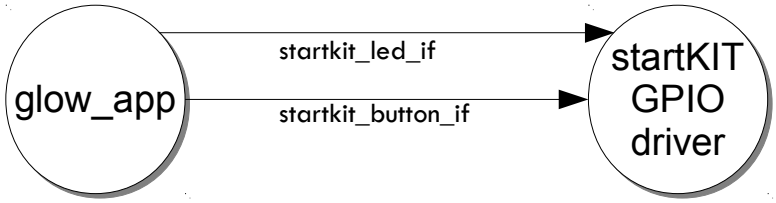

Figure 1: Glowing LEDs task diagram

2.1 The Makefile

The Makefile needs to target the startKIT. So has the line:

```
TARGET = STARTKIT
```

The startKIT support library also needs to be added to the USED_MODULES part of the Makefile:

```
USED_MODULE = lib_startkit_support
```

This will ensure that the startKIT support code is built into the application.

2.2 Application resource declaration

The resource used in the example are the ports used by the startKIT support library to access the I/O on the device. These are allocated in a structure and are always ports 32A, 4B, 4A and a single clock block:

```
startkit_gpio_ports gpio_ports =
  {XS1_PORT_32A, XS1_PORT_4B, XS1_PORT_4A, XS1_CLKBLK_1};
```

The variable gpio_ports is then used when calling the startKIT gpio driver task.

2.3 The application main() function

The `main()` function sets up two tasks running in parallel:

- `glow_app` will be the main application.
- `startkit_gpio_driver` is the driver task provided by the startKIT support library.

These tasks are connected by two interfaces that allow the application to communicate with the startKIT GPIO driver task. Details of these interfaces can be found in the startKIT support library user guide.

```
int main()
{
  // These interface connections link the application to the
  // gpio driver.
  startkit_led_if i_led;
  startkit_button_if i_button;
  par {
    on tile[0]: startkit_gpio_driver(i_led, i_button,
                                     null, null,
                                     gpio_ports);

    on tile[0]: glow_app(i_led, i_button);
  }
  return 0;
}
```

2.4 The application task

The application logic is implemented in the `glow_app` task.

```
// This function is combinable - it can share a core with other tasks
[[combinable]]
static void glow_app(client startkit_led_if leds,
                     client startkit_button_if button)
{
  timer tmr;
  int period = 1 * XS1_TIMER_HZ;      // period from off to on = 1s;
  unsigned res = 30;                  // increment the brightness in this
                                      // number of steps
  int delay = period / res;           // how long to wait between updates
  int level = 0;                      // the level of led brightness
  unsigned pattern = 0b010101010;     // the pattern output to the leds,
                                      // alternates between an X and its
                                      // inverse
  int timestamp;
  int dir = 1;

  // Take the initial timestamp of the 100Mhz timer
  tmr :> timestamp;
  while (1) {
    select {
    // After 'delay' ticks do this
    case tmr when timerafter(timestamp + delay) :> void:
      // increase the output level of the led
      level += dir * (LED_ON / res);
      if (level > LED_ON) {
        level = LED_ON;
        dir = -1;
      }
      if (level < 0) {
        level = 0;
        dir = 1;
      }
      // set the leds
      leds.set_multiple(pattern, level);
      // update the timestamp for the next timeout
      timestamp += delay;
      break;

    case button.changed():
      if (button.get_value() == BUTTON_DOWN) {
        // If the button has been pressed down then
        // invert the pattern
        pattern = ~pattern;
      }
      break;
    }
  }
}
```

The task consists of an infinite loop that repeated reacts to two events via the xC `select` statement. The first case reacts to a periodic timer and the second case reacts to a button press.

Other notes that can be seen with respect to the code are:

- The `button.changed` event is a notification from the GPIO driver tasks to indicate a button change. It is defined as part of the `starkit_button_if` interface.
- The `led.set_multiple` call tells the GPIO driver task to set the output PWM level of several LEDS in the startKIT 3x3 LED grid. It is a function defined in the `startkit_led_if` interface.

APPENDIX A - 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 startKIT.

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.

A.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 AN00174_startKIT_glowing_LEDs_demo.xe          <-- Download and execute the xCORE code
```

Once this command has executed the application will be running on the startKIT and the LEDs should flash.

A.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 and then select Run.

Once this command has executed the application will be running on the startKIT and the LEDs should flash.

APPENDIX B - 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

APPENDIX C - Full source code listing

C.1 Source code for main.xc

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#include <xs1.h>
#include <platform.h>
#include <print.h>
#include "startkit_gpio.h"

// This function is combinable - it can share a core with other tasks
[[combinable]]
static void glow_app(client startkit_led_if leds,
                     client startkit_button_if button)
{
  timer tmr;
  int period = 1 * XS1_TIMER_HZ;      // period from off to on = 1s;
  unsigned res = 30;                  // increment the brightness in this
                                      // number of steps
  int delay = period / res;           // how long to wait between updates
  int level = 0;                      // the level of led brightness
  unsigned pattern = 0b010101010;     // the pattern output to the leds,
                                      // alternates between an X and its
                                      // inverse
  int timestamp;
  int dir = 1;

  // Take the initial timestamp of the 100Mhz timer
  tmr :> timestamp;
  while (1) {
    select {
    // After 'delay' ticks do this
    case tmr when timerafter(timestamp + delay) :> void:
      // increase the output level of the led
      level += dir * (LED_ON / res);
      if (level > LED_ON) {
        level = LED_ON;
        dir = -1;
      }
      if (level < 0) {
        level = 0;
        dir = 1;
      }
      // set the leds
      leds.set_multiple(pattern, level);
      // update the timestamp for the next timeout
      timestamp += delay;
      break;

    case button.changed():
      if (button.get_value() == BUTTON_DOWN) {
        // If the button has been pressed down then
        // invert the pattern
        pattern = ~pattern;
      }
      break;
    }
```

```
  }
}


startkit_gpio_ports gpio_ports =
  {XS1_PORT_32A, XS1_PORT_4B, XS1_PORT_4A, XS1_CLKBLK_1};

// 'main' sets up the system, consisting of two tasks - one to drive
// the i/o and one to run the application that communicates with that
// driver.
int main()
{
  // These interface connections link the application to the
  // gpio driver.
  startkit_led_if i_led;
  startkit_button_if i_button;
  par {
    on tile[0]: startkit_gpio_driver(i_led, i_button,
                                     null, null,
                                     gpio_ports);

    on tile[0]: glow_app(i_led, i_button);
  }
  return 0;
}
```