Application Note: AN00177

A startKIT ADC demo

This applications provides a very simple example of using the ADC module. It uses the on-chip ADC in one shot mode (a trigger is called every 200ms from a timer) and then reads the 4 values after conversion complete notification received. It also shows an example of a select (wait on multiple events) because it also listens to the button, and lights additional LEDs when that is pressed.

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 A simple ADC example

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 and the ADC task (also supplied by the startKIT support library). The ADC task connects to a special hardware servce to access the ADC.

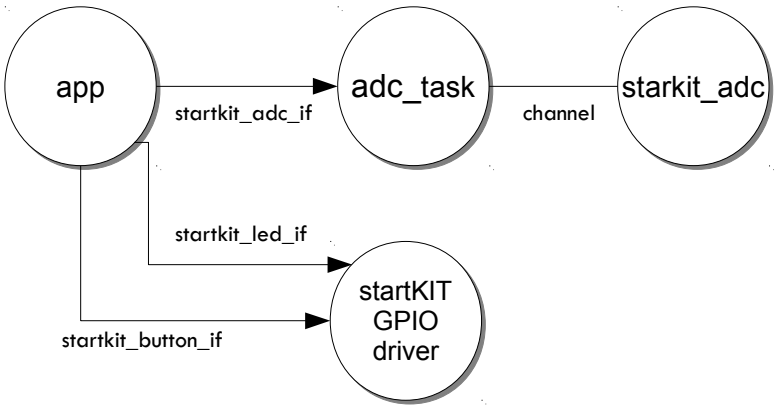

Figure 1: Glowing LEDs task diagram

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 three tasks running in parallel:

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

Note that the `startkit_adc` call in the top-level main of the program is not a software task but a hardware *service* that communicates over a channel. It also access to the ADC hardware on-chip.

These tasks are connected by three 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. The hardware service is connected to the ADC task via a channel.

```
int main()
{
  // These interface connections link the application to the GPIO task and ADC driver task
  startkit_led_if i_led;                                    //For setting LEDs
  startkit_button_if i_button;                              //For reading the button
  startkit_adc_if i_adc;                                    //For triggering/reading ADC
  chan c_adc;                                               //Used by ADC driver to connect to ADC hardware

  par {
    on tile[0].core[0]: startkit_gpio_driver(i_led, i_button,//Run GPIO task for leds/button
                                             null, null,
                                             gpio_ports);
    on tile[0].core[0]: adc_task(i_adc, c_adc, 0, adc_sample); //Run ADC server task (on same core as GPIO!)
    startkit_adc(c_adc);                                    //Declare the ADC service (this is the ADC
                                                            //hardware, not a task)
    on tile[0]: app(i_led, i_button, i_adc);                //Run the app
  }
  return 0;
}
```

2.4 The application task

The application logic is implemented in the app task.

```
void app(client startkit_led_if i_leds, client startkit_button_if i_button, client startkit_adc_if i_adc)
{
  timer t_loop;                 //Loop timer
  int loop_time;                //Loop time comparison variable

  unsigned short adc_val[4] = {0, 0, 0, 0};//ADC vals

  printstrln("App started");

  t_loop :> loop_time;          //Take the initial timestamp of the 100Mhz timer
  loop_time += LOOP_PERIOD;     //Set comparison to future time
  while (1) {
    select {

    case i_button.changed():    //Button event
      if (i_button.get_value() == BUTTON_DOWN) {
          printstrln("Button pressed!");
          i_leds.set(2, 2, LED_ON);
          i_leds.set(2, 1, LED_ON);
          i_leds.set(2, 0, LED_ON);
      }
      else {
          printstrln("Button released!");
          i_leds.set(2, 2, LED_OFF);
          i_leds.set(2, 1, LED_OFF);
          i_leds.set(2, 0, LED_OFF);
      }
      break;
                                //Loop timeout event
    case t_loop when timerafter(loop_time) :> void:
      i_adc.trigger();          //Fire the ADC!
      loop_time += LOOP_PERIOD; //Setup future time event
      break;

    case i_adc.complete():      //Notification from ADC server when aquisition complete
      i_adc.read(adc_val);      //Get the values (and clear the notfication)
      for(int i = 0; i < 4; i++){
        printstr("ADC chan ");
        printint(i);
        printstr(" = ");
        printint(adc_val[i]);
        if (i < 3) printstr(", ");
        switch (i){             //Map ADC channels to align with LEDs on startKIT
          case 0:
            i_leds.set(1, 1, adc_val[i]);
            break;
          case 1:
            i_leds.set(0, 2, adc_val[i]);
            break;
          case 2:
            i_leds.set(1, 0, adc_val[i]);
            break;
          case 3:
            i_leds.set(0, 1, adc_val[i]);
            break;
          }
        }
      printchar('\n');
      break;
    }//select
  }//while 1
}
```

The task consists of an infinite loop that repeated reacts to three events via the xC `select` statement. The first case reacts to a button press, the second to a periodic timer and the third case reacts to the completion of the ADC reading a sample.

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

- The `i_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 `i_adc.complete` event is a notification from the ADC driver tasks to indicate that the ADC has completed reading a sample. The call `i_adc.read` gets the sample value.
- The `led.set` call tells the GPIO driver task to set the output PWM level of an LED in the startKIT 3x3 LED grid. It is a function defined in the `startkit_led_if` interface.
- The `case t_loop when ...` event fires when the timer reaches the `loop_time` timeout. Every time it fires it will update the timeout making a periodic call. Each time this periodic event fires is will trigger the ADC via the `i_adc` 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 --xscope AN00177_startKIT_adc_demo.xe          <-- Download and execute the xCORE code
```

Once this command has executed the application will be running on the startKIT. Touch the ADC0..ADC3 pads/pins in the bottom left hand corner to light the LEDs. The values are also printed to the console.

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, enable **xSCOPE I/O** and then select Run.

Once this command has executed the application will be running on the startKIT. Touch the ADC0..ADC3 pads/pins in the bottom left hand corner to light the LEDs. The values are also printed to the console.

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 <xscope.h>
#include <stdlib.h>
#include "startkit_gpio.h"
#include "startkit_adc.h"

#define LOOP_PERIOD     20000000    //Trigger ADC and print results every 200ms

startkit_gpio_ports gpio_ports = {XS1_PORT_32A, XS1_PORT_4B, XS1_PORT_4A, XS1_CLKBLK_1}; //LEDs/SW, sliders,
  ↪ clock
out port adc_sample = ADC_TRIG_PORT;


void app(client startkit_led_if i_leds, client startkit_button_if i_button, client startkit_adc_if i_adc)
{
  timer t_loop;                 //Loop timer
  int loop_time;                //Loop time comparison variable

  unsigned short adc_val[4] = {0, 0, 0, 0};//ADC vals

  printstrln("App started");

  t_loop :> loop_time;          //Take the initial timestamp of the 100Mhz timer
  loop_time += LOOP_PERIOD;     //Set comparison to future time
  while (1) {
    select {

    case i_button.changed():    //Button event
      if (i_button.get_value() == BUTTON_DOWN) {
          printstrln("Button pressed!");
          i_leds.set(2, 2, LED_ON);
          i_leds.set(2, 1, LED_ON);
          i_leds.set(2, 0, LED_ON);
      }
      else {
          printstrln("Button released!");
          i_leds.set(2, 2, LED_OFF);
          i_leds.set(2, 1, LED_OFF);
          i_leds.set(2, 0, LED_OFF);
      }
      break;
                                //Loop timeout event
    case t_loop when timerafter(loop_time) :> void:
      i_adc.trigger();          //Fire the ADC!
      loop_time += LOOP_PERIOD; //Setup future time event
      break;

    case i_adc.complete():      //Notification from ADC server when aquisition complete
      i_adc.read(adc_val);      //Get the values (and clear the notfication)
      for(int i = 0; i < 4; i++){
        printstr("ADC chan ");
        printint(i);
        printstr(" = ");
        printint(adc_val[i]);
        if (i < 3) printstr(", ");
        switch (i){             //Map ADC channels to align with LEDs on startKIT
          case 0:
            i_leds.set(1, 1, adc_val[i]);
            break;
          case 1:
            i_leds.set(0, 2, adc_val[i]);
            break;
          case 2:
            i_leds.set(1, 0, adc_val[i]);
            break;
          case 3:
```

```
            i_leds.set(0, 1, adc_val[i]);
            break;
          }
        }
      printchar('\n');
      break;
    }//select
  }//while 1
}


int main()
{
  // These interface connections link the application to the GPIO task and ADC driver task
  startkit_led_if i_led;                                    //For setting LEDs
  startkit_button_if i_button;                              //For reading the button
  startkit_adc_if i_adc;                                    //For triggering/reading ADC
  chan c_adc;                                               //Used by ADC driver to connect to ADC hardware

  par {
    on tile[0].core[0]: startkit_gpio_driver(i_led, i_button,//Run GPIO task for leds/button
                                             null, null,
                                             gpio_ports);
    on tile[0].core[0]: adc_task(i_adc, c_adc, 0, adc_sample); //Run ADC server task (on same core as GPIO!)
    startkit_adc(c_adc);                                    //Declare the ADC service (this is the ADC
                                                            //hardware, not a task)
    on tile[0]: app(i_led, i_button, i_adc);                //Run the app
  }
  return 0;
}
```