XK-XMP-64 Tutorial

(VERSION 9.9)

2010/02/22

Authors:

XMOS LTD.

1 Introduction

The XK-XMP-64 is a multiprocessor board with 16 XS1-G4 devices, each running at 400 Mhz. Each device contains four cores, each core can run up to eight threads. As such, XK-XMP-64 is a 64 core board with a peak performance of 51.2 GIPS. Each core has access to 64 KBytes of RAM; the board contains 4 MBytes of memory in total. A dual Ethernet interface is available (connected to two cores) and two extension ports allow other hardware to be connected.

The XMOS originated XC language is based upon C, providing additional constructs that simplify control over I/O operations, time and concurrent behaviour. The free Software Development Tools support XC and C, allowing complete systems to be built and debugged from within a single development environment.

This tutorial provides an introduction to start developing for the XK-XMP-64, it assumes that you are familiar with C and have have seen some XC programs. In this tutorial you will:

- illuminate the LEDs on the board.
- cycle the flashing LEDs on the board.
- run a program that cycles a message around all threads.

The examples in this tutorial apply to version 9.9 of the Tools. Information on downloading, installing and using these tools is provided in the Tools User Guide.

2 Illuminate an LED

This part of the tutorial shows how to use XC ports and an output statement to illuminate the LEDs on your XK-XMP-64.

The XK-XMP-64 has 16 green LEDs. The following program illuminates a single LED on your XK-XMP-64:

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

on stdcore[60]: out port led_60 = XS1_PORT_1E;
```

```
void lightUp(out port led) {
    led <: 0x1;
    while (1) { }
}

int main() {
  par {
    on stdcore[60] : lightUp(led_60);
  }
  return 0;
}
```

The fourth line of this program declares a port variable `led_60` and initialises it with a port identifier `XS1_PORT_4F` port which originates from the `xs1.h` header file. The `1E` refers to the pin width (1 bits) and name (E). On the XK-XMP-64, the pins of 1E are connected to the 16 green LEDs. The LEDs are active high.

Ports are used to transfer data to and from the pins on the processor, thereby interfacing with external components. Integrated input and output XC statements make it easy to express I/O operations on these pins.

The first statement in `lightUp` is an output statement:

```
led <: 0x1;
```

The value specified to the right of `<:` (0b00000001) is output to the port specified to its left (`led`). This value sets the LED pin high, causing the corresponding LED to illuminate. Outputting a 0 value to `led` causes the led to go dark.

The infinite loop introduced after the output statement prevents the program terminating, ensuring that the LED remains illuminated.

Ports must be declared as global variables. The optional `out` qualifier allows the compiler to check for correct usage, thereby helping to reduce programming errors.

Compile and run this program on your XK-XMP-64. (See the Tools User Guide for information on using the Tools, use the XK-XMP-64 as your target. Please note that building binaries for the XK-XMP-64 may take a few minutes.) The LED on the left near the middle of the board should illuminate.

Modify the core numbers to illuminate other LEDS.

3 Flash all LEDs

The XS1-G4 devices are connected in a hypercube network. This part of the tutorial shows you how to use channels and timers to flash all LEDs on the XK-XMP-64 sequentially.

Timers are a special type of port that, when input from, return the current time. Timers provide a view onto a 100 MHz reference clock, and can be used to determine when an event happens or to delay execution until a particular time.

The following code declares a timer named `tmr` and then inputs the time into the variable `t`:

```
timer tmr;
int t;
tmr :> t;
```

After the current time has been recorded, you can increment the time and then delay a following input until after this time is reached:

```
t += FLASH_PERIOD;
tmr when timerafter(t) :> void;
```

The processor must complete an input operation once a condition is met, even if the input value is not required. This is expressed in XC as an input to `void`. This code sequence can be used to delay an output operation to the LED pins which, when executed in a loop, flashes the LED.

Channels provide a communication mechanism between two processes. A channel is declared as a `chan`, in this example we declare an array of 16 channels. Each channel has two ends, and can hence be passed to two processes. In this example, each process on core $4 \times k$ gets channel ends k and $k + 1$. It treats channel end k as an input channel, and $k + 1$ as an output channel. In this example, we pass a token through this ring of channel, when a process receives the token it will change the led value, wait, and pass the token on.

The function `lightUp` lights a led, waits, passes the delay onto the next process, and then waits to receive a delay, before switching the led off, timing, etc. In order to make sure that only one process is switching its led at any one time, all processes except the one on node 0 start by waiting for the token.

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

on stdcore[0]: out port led_0 = XS1_PORT_1E;
on stdcore[4]: out port led_4 = XS1_PORT_1E;
on stdcore[8]: out port led_8 = XS1_PORT_1E;
on stdcore[12]: out port led_12 = XS1_PORT_1E;
on stdcore[16]: out port led_16 = XS1_PORT_1E;
on stdcore[20]: out port led_20 = XS1_PORT_1E;
on stdcore[24]: out port led_24 = XS1_PORT_1E;
on stdcore[28]: out port led_28 = XS1_PORT_1E;
on stdcore[32]: out port led_32 = XS1_PORT_1E;
on stdcore[36]: out port led_36 = XS1_PORT_1E;
on stdcore[40]: out port led_40 = XS1_PORT_1E;
on stdcore[44]: out port led_44 = XS1_PORT_1E;
on stdcore[48]: out port led_48 = XS1_PORT_1E;
on stdcore[52]: out port led_52 = XS1_PORT_1E;
on stdcore[56]: out port led_56 = XS1_PORT_1E;
on stdcore[60]: out port led_60 = XS1_PORT_1E;

void lightUp(out port led, int number, chanend inC, chanend outC) {
  timer tmr;
  int t, ledStatus = 1;

  if (number == 0) {
    inC :> number;
  }
  while (1) {
    led <: ledStatus;
    tmr :> t;
    t += number;
    tmr when timerafter(t) :> void;
    outC <: number;
    inC :> number;
    ledStatus = !ledStatus;
  }
}

int main() {
  chan c[16];
  par {
    on stdcore[0] : lightUp(led_0, 50000000, c[0], c[1]);
    on stdcore[4] : lightUp(led_4, 0, c[1], c[2]);
    on stdcore[8] : lightUp(led_8, 0, c[2], c[3]);
    on stdcore[12] : lightUp(led_12, 0, c[3], c[4]);
```

```
    on stdcore[16] : lightUp(led_16, 0, c[4], c[5]);
    on stdcore[20] : lightUp(led_20, 0, c[5], c[6]);
    on stdcore[24] : lightUp(led_24, 0, c[6], c[7]);
    on stdcore[28] : lightUp(led_28, 0, c[7], c[8]);
    on stdcore[32] : lightUp(led_32, 0, c[8], c[9]);
    on stdcore[36] : lightUp(led_36, 0, c[9], c[10]);
    on stdcore[40] : lightUp(led_40, 0, c[10], c[11]);
    on stdcore[44] : lightUp(led_44, 0, c[11], c[12]);
    on stdcore[48] : lightUp(led_48, 0, c[12], c[13]);
    on stdcore[52] : lightUp(led_52, 0, c[13], c[14]);
    on stdcore[56] : lightUp(led_56, 0, c[14], c[15]);
    on stdcore[60] : lightUp(led_60, 0, c[15], c[0]);
  }
  return 0;
}
```

Compile and run this program on your XK-XMP-64. All 16 green leds should go on in order, and go off in order. Note that the nodes are numbered according to their position in the hypercube, as shown in Figure 1

Modify this program to change the speed at which the LED flashes; make the leds on the edge flash slow and the ones in the center flash fast.

4 Using replicators

You can use a replicator to easily place communicating processes on all cores. For example:

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

void pipelineStep(chanend inC, chanend outC, int number) {
  if (number == 0) {
    inC :> number;
  }
  while (1) {
    number = number + 1;
    outC <: number;
    inC :> number;
  }
}
```

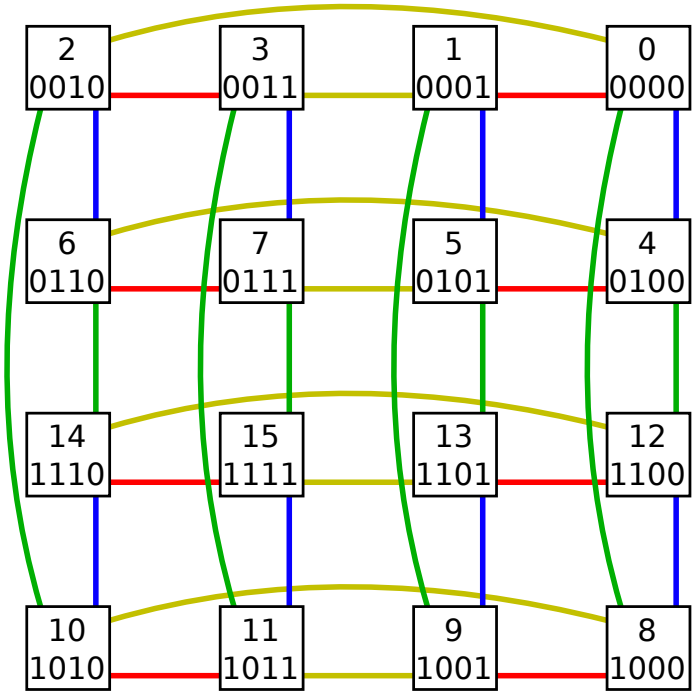

Figure 1 *XK-XMP-64 Interconnection: red links connect dimension 0, yellow links dimension 1, blue links dimension 2, and green links connect dimension 3.*

```
int main() {
  chan c[512];
  par (int i = 0; i < 512; i++) {
    on stdcore[i/8] : pipelineStep(c[i], c[(i+1)&511], i);
  }
  return 0;
}
```

When you break the program with an `xrun --dumpstate` you can see the state of all threads.

5 XK-XMP-64 Board Layout and Further Reading

Information on pin/port mappings and the initializers to use to access features of the XK-XMP-64 board, see the XK-XMP-64 Hardware Manual.

Further information on the XC language can be found in Programming XC for XMOS Devices.

Information on the tools is available in the Tools User Guide.

Information on the XS1 architecture is available in The XS1 Architecture, XS1-G System and XS1 Assembly Language Manual documents.

6 Document History

Date	Release	Comment
2010/02/22	9.9	First release.