XC-2 Development Board Tutorial

IN THIS DOCUMENT

1 Introduction

The XC-2 Ethernet Kit is a rapid and cost effective route for developing Ethernet-based products such as audio/video bridging applications and industrial control systems. The XC-2 comprises a single XS1-G4 device, 10/100-BASE-T Ethernet PHY, 4Mbits SPI flash memory, 10 LEDs and two press-buttons. I/O expansion areas are provided for connecting additional components, and an XSYS debugging interface.

This tutorial shows you how to write some simple XC programs that control and respond to the XC-2 board components. In this tutorial you learn how to:

- illuminate an LED on the board
- flash an LED at a fixed rate
- send the message “Hello World”’ to your PC over a serial link
- create multiple concurrent threads that flash LEDs at different rates
- send a token between multiple threads, each flashing an LED in sequence
- add a button listener thread that changes the token direction
- ping the IP address of your XC-2 from a PC, causing it to flash an LED

2 Illuminate an LED

This part of the tutorial shows you how to illuminate an LED on your XC-2, using an XC port and an output statement.

Publication Date: 2012/2/29 Document Number: X8118A

2.1 Create an application

2.2 Add the application code

The program below illuminates an LED on an XC-2.

```
#include <xs1.h>

out port x0ledb = XS1_PORT_1J;

int main () {
  x0ledb <: 1;
  while (1)
    ;
  return 0;
}
```

To add this code to your application, follow these steps:

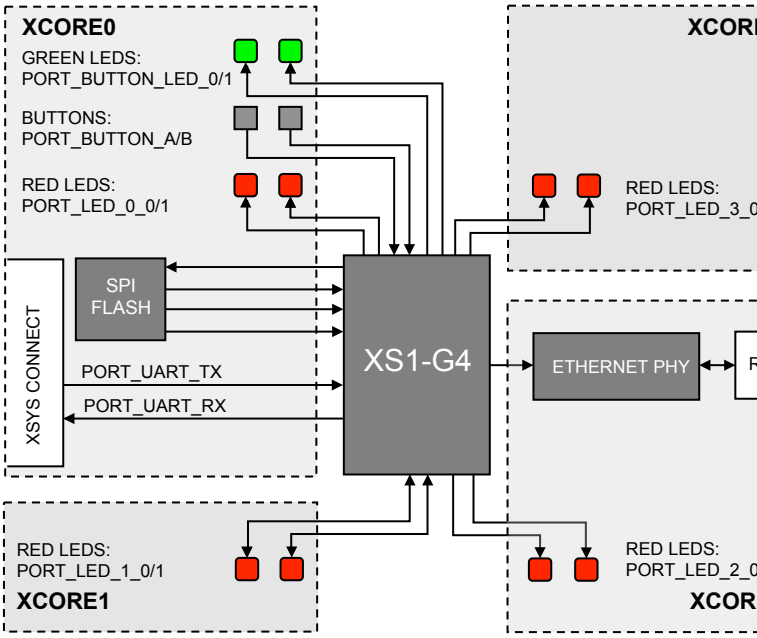
1. Click **Select All** in the code window above to highlight the code.
2. Click and hold the highlighted code, drag the cursor to the editor and release. The XDE copies this code into the editor.
3. Choose **File ▶ Save** () to save your changes to file.

2.3 Examine the application code

Take a look at the application code in the editor. The declaration

```
out port x0ledb = XS1_PORT_1J;
```

declares an output port named `x0ledb`, which refers to the 1-bit port 1J. On the XC-2, the I/O pin of port 1J is connected to the LED next to Button B.



Show image of port map..

XC input and output statements make it easy to express I/O operations on ports. The statement

```
x0ledb <: 1;
```

causes the value specified to the right of `<:` to be output to the port specified to its left (`x0ledb`). The port then drives the LED high causing the LED to illuminate green.

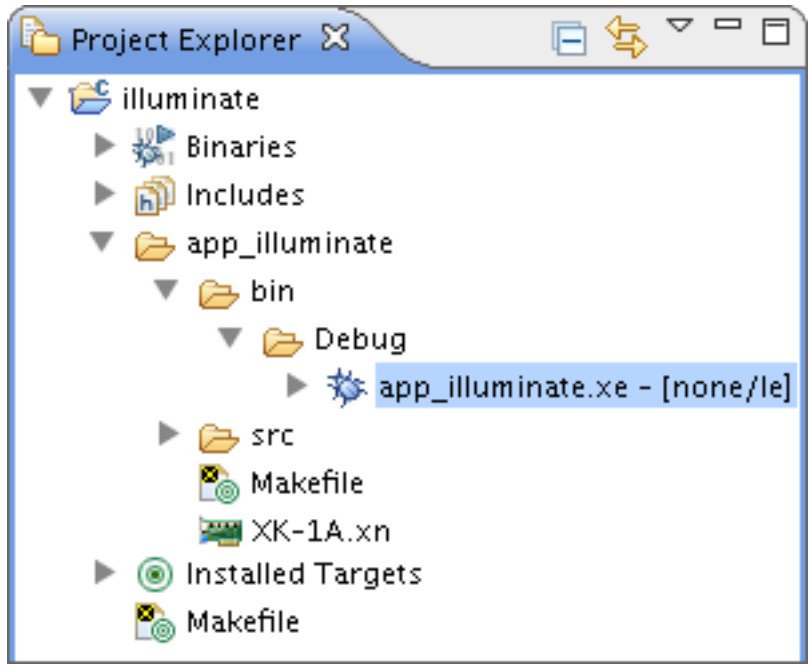
The empty `while` loop prevents the program from terminating, which ensures that the LED remains illuminated.

2.4 Build and run your application

To build and run your application, follow these steps:

1. In the **Project Explorer**, click your project to select it, and then choose the menu option **Project ▶ Build Project** ( ).

   The XDE displays its progress in the **Console**. When the build is complete, the XDE adds the compiled binary file to the application subfolder `bin/Debug`.



2. Choose **Run ▶ Run Configurations**.
3. In the **Run Configurations** dialog, in the left panel, double-click **XCore Application**.
4. In the right panel, in **Name**, enter the name `illuminate`.
5. In **Project**, ensure that your project is displayed. If not, click **Browse** to open the **Project Selection** dialog, select your project, and then click **OK**.
6. In **C/C++ Application**, click **Search Project** to open the **Program Selection** dialog, select your application binary, and then click **OK**.
7. In **Device options**, in **Run on**, select the option **hardware**, and in **Target**, ensure that the option "XMOS XC-2 Board" is selected.

   If your hardware is not displayed, ensure that your XC-2 is connected to your PC, and then click **Refresh list**.
8. Click **Run** to save your configuration and run it.

   The XDE loads the application binary onto your XC-2, displaying its progress in the **Console**. When the binary is loaded, the **Console** is cleared.
9. On your XC-2, verify that the LED is illuminated green.
10. In the **Console**, click the **Terminate** button () to stop your application running.

3 Flash an LED

This part of the tutorial shows you how to flash an LED at a fixed rate, using an XC timer and an input statement.

3.1 Create a second application

The XDE lets you work with multiple applications at the same time. You can group related applications into a single project, making them easy to organize and move around together.

To create a second application in your existing project, follow these steps:

3.2 Add the application code

The program below flashes a single LED on an XC-2.

```
#include <xs1.h>

#define FLASH_PERIOD 20000000

out port x0ledb = XS1_PORT_1J;

int main (void) {
  timer tmr;
  unsigned ledOn = 1;
  unsigned t;
  tmr :> t;
  while (1) {
    x0ledb <: ledOn;
    t += FLASH_PERIOD;
    tmr when timerafter (t) :> void;
    ledOn = !ledOn;
  }
  return 0;
}
```

To add this code to your application, follow these steps:

1. Click **Select All** in the code window above to highlight the code.
2. Right-click in the code window and select **Copy** from the pop-up menu.
3. Right-click in the editor window and select **Paste** from the pop-up menu.
   The XDE copies this code into the editor.
4. Choose **File ▶ Save** () to save your changes to file.

3.3 Examine the application code

Take a look at the application code in the editor. The declaration

```
timer tmr;
```

declares a variable named `tmr`, and allocates an available hardware timer. The L1 provides 10 timers, which can be used to determine when an event happens, or to delay execution until a particular time. Each timer contains a 32-bit counter that is incremented at 100MHz and whose value can be input at any time.

The statement

```
tmr :> t;
```

inputs the value of the `tmr` counter into the variable `t`. Having recorded the current time, the statement

```
t += FLASH_PERIOD;
```

increments this value by the required delay, and the statement

```
tmr when timerafter(t) :> void;
```

delays inputting a value until the specified time is reached. The input value is not needed, which is expressed as an input to `void`.

3.4 Build and run your application

To build and run your application, follow these steps:

1. In the **Project Explorer**, click your project to select it, and then choose the menu option **Project ▶ Build Project** ().

   The XDE builds **all** applications in your project, displaying its progress in the **Console**. When the build is complete, the XDE adds the compiled binary file to the application subfolder `bin/Debug`.

2. Create a new Run Configuration for your application named `flash`, and run it.

   Show reminder..

   Follow these steps:

3. Choose **Run ▶ Run Configurations**.

4. In the **Run Configurations** dialog, in the left panel, double-click **XCore Application**.

5. In the right panel, in **Name**, enter the name `flash`.

6. In **Project**, ensure that your project is displayed. If not, click **Browse** to open the **Project Selection** dialog, select your project, and then click **OK**.

7. As there are now two applications in your project, the XDE is unable to select one automatically. To select, in **C/C++ Application**, click **Search Project** to open the **Program Selection** dialog, select your application binary, and then click **OK**.

8. In **Device options**, in **Run on**, select the option **hardware**, and in **Target**, ensure that the option "XMOS XC-2 Board" is selected.

9. Click **Run** to save your configuration and run it.

   The XDE loads the application binary onto your XC-2, displaying its progress in the **Console**. When the binary is loaded, the **Console** is cleared.

10. On your XC-2, verify that the LED is flashing on-off, and then click the **Terminate** button () to stop your application running.

3.5 Switch between applications

The **Run** button () can be used to switch between applications. To complete this part of the tutorial, follow these steps:

1. Click the arrow to the right of the **Run** button and select the Run Configuration named `illuminate`.

2. On your XC-2, verify that the LED is illuminated.

3. Click the arrow to the right of the **Run** button and select the Run Configuration named `flash`.

4. On your XC-2, verify that the LED is flashing on-off.

5. Modify the source of the flashing LED application to change the value of `FLASH_PERIOD` from 20000000 to 40000000.

6. To build and run, just click the **Run** button.

   The XDE launches the Run Configuration you most recently selected.

7. On your XC-2, verify that the LED is flashing on-off at half the rate it was flashing previously, and then click the **Terminate** button () to stop your application running.

3.6 Disable an application from the build

By default when you build or run an application, the XDE checks whether the source code has changed in **any** application in your project and rebuilds all applications that have changed. On some systems, the time taken to inspect all folders can produce a noticable increase in the build time.

During development, you can control which applications are included in the build. To disable an application from the build, open the project-level `Makefile` in the editor and uncheck the application from the build list.

3.7 Rename your project

Now that you have two applications in your project, you may wish to rename your project to represent the fact that it is a container for a collection of applications. To complete this part of the tutorial, follow these steps:

1. In the **Project Explorer**, right-click on your project and select **Rename**.
2. In the **Rename Resource** dialog, enter the name `xc-2 examples` and click **OK**.
3. Verify that your project is renamed.

4 Interface with a host over a serial link

This part of the tutorial shows you how to implement a UART protocol that transmits a message from the XS1-G4 to your PC over a serial link. The XC-1A has a chip that performs a USB-to-serial conversion. When the board is connected to a PC using a USB cable, this chip presents a virtual COM port that can be interfaced using a terminal emulator. (Currently on MACs, the virtual COM port cannot be supported at the same time as the JTAG interface, preventing you from completing the following exercise.)

4.1 Create an application

The program below serializes a byte of data and transmits its individual bits over a 1-bit port using the UART transmission protocol:

```
#include <platform.h>

#define BIT_RATE 115200
#define BIT_TIME XS1_TIMER_HZ / BIT_RATE

void txByte(out port TXD, int byte) {
  unsigned time;
  timer t;

  /* input initial time */
  t :> time;

  /* output start bit */
  TXD <: 0;
  time += BIT_TIME;
  t when timerafter(time) :> void;

  /* output data bits */
  for (int i=0; i<8; i++) {
    TXD <: >> byte;
    time += BIT_TIME;
    t when timerafter(time) :> void;
  }

  /* output stop bit */
```

```
  TXD <: 1;
  time += BIT_TIME;
  t when timerafter(time) :> void;
}
```

Before continuing to the next part of this tutorial, create a new application using this code.

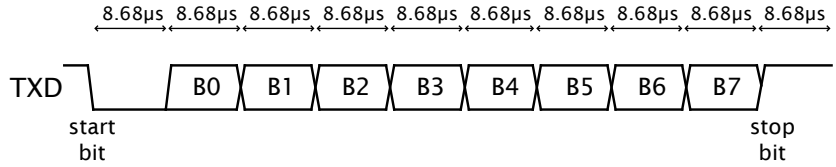
Show reminder..

Follow these steps:

1. Choose **File ▶ New ▶ XDE Application** ( ).
2. In the **New Application** dialog, in **Application Name**, enter a name for the application.
3. In **Project**, ensure that your project is selected.
4. In **Target Hardware**, select the option **XC-2 Development Board**.
5. In **Application Sofware**, select the option **Empty XC File**.
6. Click **Finish** to create an empty source file.
7. Copy and paste the code in the window above into your new source file, and save.

4.2 Examine the application code

A UART translates data between parallel and serial forms for transmission over a serial link. Each bit of data is driven for a fixed period, during which time the receiver must sample the data. The diagram below shows the transmission of a single byte of data at a rate of 115200 bits/s, which means that each bit is driven for 8.68us.



The function `txByte` outputs a byte by first outputting a start bit, following by a conditional input on a timer that waits for the bit time to elapse; the data bits and stop bit are output in the same way.

The output statement in the `for` loop

```
TXD <: >> byte;
```

includes the modifier >>, which right-shifts the value of `byte` by the port width (1 bit) after outputting the least significant port-width bits. This operation is performed in the same instruction as the output, making it more efficient than shifting the value as a separate operation afterwards.

4.3 Exercise

To complete this part of the tutorial, perform the following tasks:

1. Load a terminal emulator program on your PC and connect it to the virtual COM port provided by the XC-2. A simple terminal emulator is available from the *XCore community website<http:/www.xcore.com/tag_search?tag=uart>_*
2. Complete the program above by declaring a port for the UART and by writing a `main` function that outputs the message `Hello World!` to this port.

   Show a tip..

   You can find the relevant port in the XC-1A Hardware Manual.
3. Build your application, create a new Run Configuration, and run it.
4. Verify that the terminal receives and displays the message, and then click the **Terminate** button () to stop your application running.

5 Run tasks concurrently

This part of the tutorial shows you how to flash multiple LEDs driven by different processors on your XC-2, using the xc `par` and `on` statements.

5.1 Create an application

The program below creates four concurrent threads, each running an instance of a function that flashes an LED:

```
#include <platform.h>

#define FLASH_PERIOD 20000000

on stdcore[0] : out port x0ledB = PORT_LED_0_1;
on stdcore[0] : out port x0ledA = PORT_LED_0_0;
on stdcore[1] : out port x1ledB = PORT_LED_1_1;
on stdcore[1] : out port x1ledA = PORT_LED_1_0;

void flashLED(out port led, int delay);

int main(void) {
  par {
    on stdcore[0]: flashLED(x0ledB, FLASH_PERIOD);
    on stdcore[0]: flashLED(x0ledA, FLASH_PERIOD);
    on stdcore[1]: flashLED(x1ledB, FLASH_PERIOD);
    on stdcore[1]: flashLED(x1ledA, FLASH_PERIOD);
```

```
  }
  return 0;
}
```

5.2 Examine the application code

Take a look at the application code in the editor.

The header file `platform.h` provides a declaration of the global variable `stdcore`, which can be used to specify the placement of port declarations and threads.

The `par` statement provides a simple way to create *concurrent threads* that can run independently of one another. The `on` statement instructs the compiler on which processor each port is connected and each thread is executed.

An‘‘on‘‘ statement may only be used with threads created by `main`, in which case `main` may contain only channel declarations, a single `par` statement and an optional `return` statement.

5.3 Exercise 1

To complete this part of the tutorial, perform the following tasks:

1. Modify the code from Section sec:flash to form the body of the function `flashLED`.
2. Compile and run this program on your XC-2.

   Four LEDs on one side of the XS1-G4 should continually flash at a fixed rate.
3. In the **Console**, click the **Terminate** button () to stop your application running.

5.4 Exercise 2

Experiment with different period values for each of the threads so that the threads can be seen to be operating independently of one another.

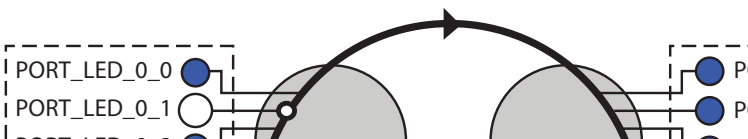
5.5 Exercise 3

Extend this program to flash the LEDs on both sides of the XS-G4.

Note that the LEDs on processor 3 are in reverse order on the XC-2.

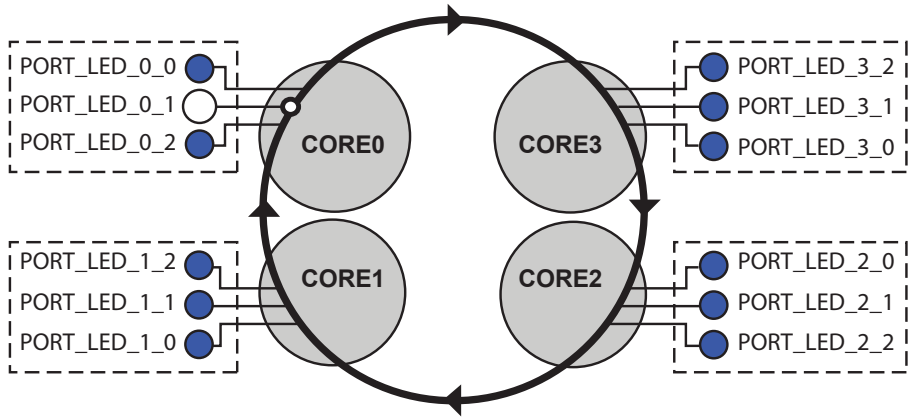
6 Flash multiple LEDs in sequence

This part of the tutorial shows you how to use XC channels to flash eight of the LEDs on your XC-2 in the round robin sequence illustrated below:



6.1 Create an application

The function below implements a component of the token ring illustrated above, repeatedly inputting a token from its left neighbor, flashing an LED and outputting the token to its right neighbor.




```
void tokenFlash(chanend left, chanend right,
                out port led, int delay, int isMaster) {
  timer tmr;
  unsigned t;
  if (isMaster)     / * master inserts token into ring */
    right <: 1;
  while (1) {
    int token;
    left :> token;   /* input token from left neighbor */
    led <: 1;
    tmr :> t;
    tmr when timerafter(t+delay) :> void;
    led <: 0;
    right <: token;  /* output token to right neigbor */
  }
}
```

6.2 Examine the application code

The first two function parameters are channel ends, the third an LED port and the fourth a Boolean value indicating whether or not the thread executing the function is the designated *master*. The master inserts a token into the ring.

The xc input and output statements are used to communicate the token between threads. As channels are synchronous, each output operation blocks until a matching input operation is ready, ensuring that precisely one thread has possession of the token at any time.

The function below constructs part of the token ring:

```
int main(void) {
  chan c0, c1, c2, ...;
  par {
    on stdcore[0]: tokenFlash(c0, c1, x0ledA, PERIOD, 1);
    on stdcore[0]: tokenFlash(c1, c2, x0ledB, PERIOD, 0);
    ...
  }
  return 0;
}
```

A channel is declared using the keyword `chan`. The locations of its two channel ends are established through its use in two statements of the `par`.

A total of eight channels are required to complete this program, each of which must be used in two threads: once as a left argument and once as a right argument to the function `tokenFlash`.

6.3 Exercise

Complete this program, compile and run it on your XC-2.

As the token cycles around the eight threads, the eight LEDs should each flash in sequence.

7 Add a button controller to the token ring

This part of the tutorial shows you how to detect a button press and respond to it, using the XC `select` statement.

7.1 Create an application

A `select` statement is used to respond to one of a set of inputs, depending on which becomes ready first. If more than one of input becomes ready at the same time, only one is executed.

The function below waits for either a token to be received, in which case it passes it on, or for a button to be pressed, in which case it holds the token when received until the button is pressed again:

```
void buttonListener(chanend left, chanend right, port button) {
  int token;
  while (1)
    select {
      case left :> token :
        /* pass token on */
        right <: token;
        break;
      case button when pinsneq(0xf):> void :
```

```
        /* wait for token, then hold until button is pressed */
        left :> token;
        button when pinsneq(0xf):> void; /* push down */
        button when  pinseq(0xf):> void; /* release */
        right <: token;
        break;
      }
}
```

The guarded input statement:

```
case left :> token :
```

becomes ready when the token arrives, in which case it is input and passed to the next thread. The guard:

```
case button when pinsneq(0xf):> void :
```

becomes ready when the value on the pins connected to the port `button` is not equal to the bit pattern `0xf`. This signifies that the button was pressed. The body of this case then waits for the button to be pressed again and released before continuing.

7.2 Exercise 1

Add the function `buttonListener` to the token ring from the previous section that responds to either one of the two press-buttons. You can find their ports in the diagram at the end of this tutorial on page sec:portmap.

Compile and run this program on your XC-2.

Verify that the LED stops cycling after a full cycle when one of the buttons is pressed. Pressing the button again causes the next cycle to commence.

7.3 Exercise 2

Modify the program so that pressing a button changes the cycle direction.

The simplest solution is to add a `select` statement to the function “flashLED‘ that inputs from either its left or right neighbor, and extend `buttonListener` similarly.

8 Ping your XC-2 from a PC over its Ethernet connection

This part of the tutorial shows you how to integrate the XMOS Ethernet controller into your own design. The 10/100-BASE-T Ethernet interface on the XC-2 is implemented using a standard physical layer interface device. The MAC and MII level protocols are implemented in software.

8.1 Download the Ethernet software

The software can be downloaded as a ZIP file from http://www.xmos.com/xc2/, and imported directly into the XDE as a pre-configured project (for instructions see the Tools User Guide) or uncompressed and built using the provided Makefile.

Open the file `test.xc` and change the preprocessor value `OWN_IP_ADDRESS` to an IP address that your network can route.

Compile and run this program on your XC-2.

You should be able to ping and receive a response from the board.

Note that you may need to disable your firewall.

8.2 Exercise

Modify the function `demo` in the file `test.xc` so that it flashes one of the green LEDs each time it receives a ping request.

9 What to read next

Congratulations, you're now ready to start building Ethernet-based products using XC and your XC-2. For information on how to incorporate this software into your own design, refer to the files `README.txt` and `API.txt`.