XC-1A Development Board Tutorial

IN THIS DOCUMENT

1 Introduction

The XC-1A is a low-cost development board based on the XMOS XS1-G4 device. It includes a single G4 device, 4Mbits SPI FLASH memory, 16 user-configurable LEDs, four push buttons, a speaker, JTAG and serial interfaces, four expansion areas suitable for IDC headers and a through-hole prototyping area for connecting external components..

This tutorial shows you how to write some simple XC programs that control and respond to the XC-1A board components. In this tutorial you learn how to:

- illuminate an LED on the board
- ash an LED at a xed 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 LED color

2 Illuminate an LED

This part of the tutorial shows you how to illuminate an LED on your XC-1A, using an XC port and an output statement.

2.1 Create a project

2.2 Add the code

The program below illuminates an LED on an XC-1A.

Publication Date: 2012/11/8 Document Number: X3948A

```
#include <xs1.h>

out port bled = XS1_PORT_4C;

int main () {
  bled <: 0b0001;
  while (1)
    ;
  return 0;
}
```

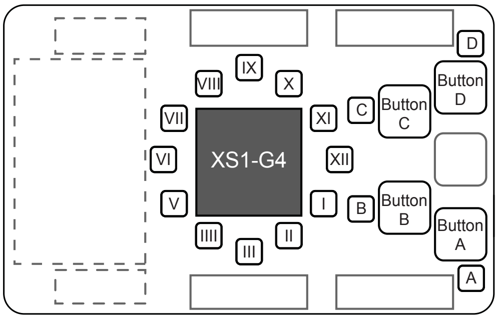
Copy and paste the code into your project, and then choose **File ▶ Save** () to save your changes to file.

2.3 Examine the code

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

```
out port bled = XS1_PORT_4C;
```

declares an output port named `bled`, which refers to the 4-bit port 4C. On the XC-1A, the I/O pins of port 4C are connected to the LEDs positioned next to the four press-buttons (collectively referred to as button-LEDs) that contain green diodes.



Show image of port map..

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.

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

```
bled <: 0b0001;
```

causes the value specified to the right of `<:` to be output to the port specified to its left (`led`). The port then drives LED next to button A high and the other LEDs low, causing the LED to illuminate green and the other LEDs to remain off.

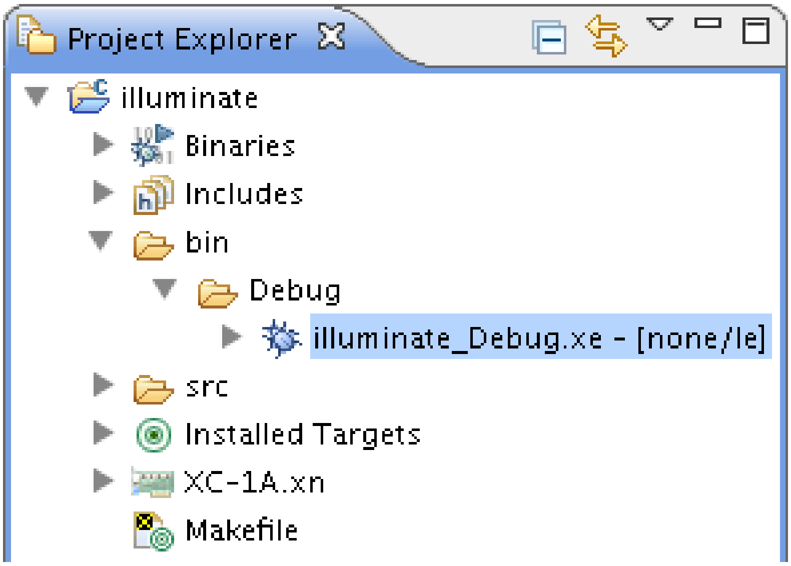
The empty `while` loop prevents the program from terminating, which ensures that the LED remains illuminated.

2.4 Build and run your project

To build and run your project, 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 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-1A Board” is selected.

   If your hardware is not displayed, ensure that your XC-1A is connected to your PC, and then click **Refresh list**.

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

   The XDE loads the binary onto your XC-1A, displaying its progress in the **Console**. When the binary is loaded, the **Console** is cleared.

9. On your XC-1A, verify that BUTTONLED A is illuminated green.

10. In the **Console**, click the **Terminate** button () to stop your application running.

2.5 Exercise

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

1. Modify your code so that it illuminates all four BUTTONLEDs.

   You should change the value output to the port `bled` so that all four LEDs are driven high.

```
#include <xs1.h>

out port led = XS1_PORT_4C;

int main () {
  bled <: 0b1111;
  while (1)
    ;
  return 0;
}
```

2. Click the **Run** button (▶) to reload your last Run Configuration.

   The XDE determines that your source code has been updated and re-builds it, displaying progress in the **Console**.

   If your code contains errors, the XDE displays a dialog asking if you want to continue launching the application. Click **No**, locate the first error in the **Console** and double-click it to go to the offending line in the editor. When you have fixed all errors, re-run your application.

3. On your XC-1A, verify that all four BUTTONLEDs are illuminated, and then 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 new project

3.2 Add the application code

The program below flashes a single LED on an XC-1A.

```
#include <xs1.h>

#define FLASH_PERIOD 20000000

out port bled = XS1_PORT_4C;

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

Copy and paste the code into your project, and then choose **File ▶ Save** () to save your changes to file.

3.3 Examine the code

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

```
timer tmr;
```

declares a variable named `tmr`, and allocates an available hardware timer. Each core on the G4 device 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 `tmr`'s 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 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 project 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. In **Device options**, in **Run on**, select the option **hardware**, and in **Target**, ensure that the option "XMOS XC-1A Board" is selected.
8. Click **Run** to save your configuration and run it.

   The XDE loads the binary onto your XC-1A, displaying its progress in the **Console**. When the binary is loaded, the **Console** is cleared.

9. On your XC-1A, verify that BUTTONLEDA is flashing on-off, and then click the **Terminate** button () to stop your application running.

3.5 Switch between projects

The **Run** button () can be used to switch between projects. 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-1A, verify that all four LEDs are illuminated.
3. Click the arrow to the right of the **Run** button and select the Run Configuration named `flash`.
4. On your XC-1A, verify that BUTTONLEDA 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-1A, verify that BUTTONLEDA 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 Exercise

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

1. Modify your flashing LED application so that both BUTTONLEDA and BUTTONLEDB are flashed, with BUTTONLEDA flashed twice as fast as BUTTONLEDB.

   Show a tip..

   You should output the following pattern to the port bled: 0b0011, 0b0010, 0b0011, 0b0010, 0b0001, 0b0000, 0b0001 and 0b0000.

   Explain the solution in more detail..

   You can define an array of integers an initialize it with the values 0b0011, 0b0010, 0b0011, 0b0010, 0b0001, 0b0000, 0b0001 and 0b0000. Then use a loop to output this sequence of values to the port 4C.

   Show a sample answer..

```
#include <xs1.h>

#define FLASH_PERIOD 20000000

out port bled = XS1_PORT_4C;

int pattern[] = {0b0011,
                 0b0010,
                 0b0011,
                 0b0010,
                 0b0001,
                 0b0000,
                 0b0001,
                 0b0000};

int main (void) {
  timer tmr;
  unsigned t;
  unsigned i = 0;
  tmr :> t;
  while (1) {
    t += FLASH_PERIOD;
    tmr when timerafter (t) :> void;
    bled <: pattern[i];
    i = (i+1) % 8;
  }
  return 0;
}
```

2. Run your application.
3. On your XC-1A, verify that the two LEDs are flashing at different speeds, and then click the **Terminate** button () to stop your application running.

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 a project

The program below inputs from one of two timers in a loop.

```
#include <xs1.h>
#define BIT_RATE 115200
#define BIT_TIME XS1_TIMER_HZ / BIT_RATE
out port TXD = XS1_PORT_1H;

int main(){
  return 0;
}

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 project using this code.

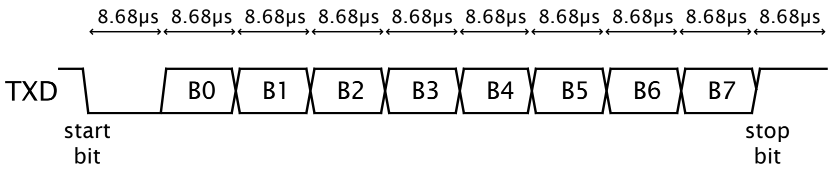
Show reminder..

Follow these steps:

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

4.2 Examine the 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:



The quiescent state of the link is high. A byte is sent by first driving a start bit (0), followed by the data bits and then a stop bit (1). A rate of 115200 bits/s means that each bit is driven for 1/115200 = 8:68us.

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

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 steps:

1. Load a terminal emulator program on your PC and connect it to the virtual COM port provided by the XC-1A. A simple terminal emulator is available from the XMOS community website.
2. Complete the program by declaring a port for the UART and by writing a `main` function that outputs the message `Hello World!` to this port.

   You can find the relevant port in the XC-1A Hardware Manual[1].
3. Run your application.
4. The terminal should receive and display the message.

5 Flash and cycle LEDs at different rates

This part of the tutorial shows you how to flash multiple CLOCKLEDs concurrently on your XC-1A.

5.1 Create an application

The program below flashes a single CLOCKLED.

```
#include <platform.h>
#define PERIOD 20000000

out port cled0 = PORT_CLOCKLED_0;
out port cled1 = PORT_CLOCKLED_1;
out port cled2 = PORT_CLOCKLED_2;
out port cled3 = PORT_CLOCKLED_3;
out port cledG = PORT_CLOCKLED_SELG;
out port cledR = PORT_CLOCKLED_SELR;

void flashLED (out port led, int period);

int main (void) {
  par {
    on stdcore [0]: { cledG <: 1;
                      flashLED (cled0 , PERIOD);
                    }
```

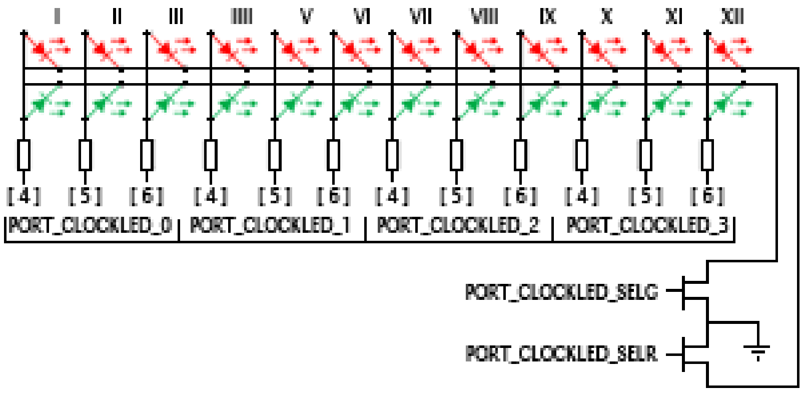
[1] http://www.xmos.com/published/xc1ahw

```
    on stdcore [1]: flashLED (cled1, PERIOD);
    on stdcore [2]: flashLED (cled2, PERIOD);
    on stdcore [3]: flashLED (cled3, PERIOD);
  }
  return 0;
}

void flashLED (out port led, int period){
}
```

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

5.2 Examine the application code

The schematic for the 12 clock-LEDs is shown below; the location of the ports are shown in the XC-1A Hardware Manual[2].



The LED anodes are connected to four 8-bit ports: PORT_CLOCKLED_0 on XCore 0, PORT_CLOCKLED_1 on XCore 1, PORT_CLOCKLED_3 on XCore 3 and PORT_CLOCKLED_4 on XCore 4. The LED cathodes are connected to two 1-bit ports on XCore 0: PORT_CLOCKLED_SELG (green) and PORT_CLOCKLED_SELR (red). This means that two pins must be driven to illuminate a clock-LED.

The `par` statement provides a simple way to create concurrent threads that 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.

The program creates four concurrent threads, each running an instance of a function `flashLED`.

[2]http://www.xmos.com/published/xc1ahw

5.3 Exercise 1

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

1. Implement the body of the function flashLED so that it flashes a single LED. Note that the pins are connected to bits 4–6 on the 8-bit port.
2. Build your application, create a new Run Configuration, and run it.
3. On your XC-1A, verify that four LEDs on the clockface flash continually at a fixed rate, and then click the **Terminate** button () to stop your application running.

5.4 Exercise 2

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

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

6 Run tasks concurrently

This part of the tutorial shows you how to use XC channels to flash eight of the LEDs on your XC-1A in the round robin sequence..

6.1 Create an application

The program below flashes a single CLOCKLED.

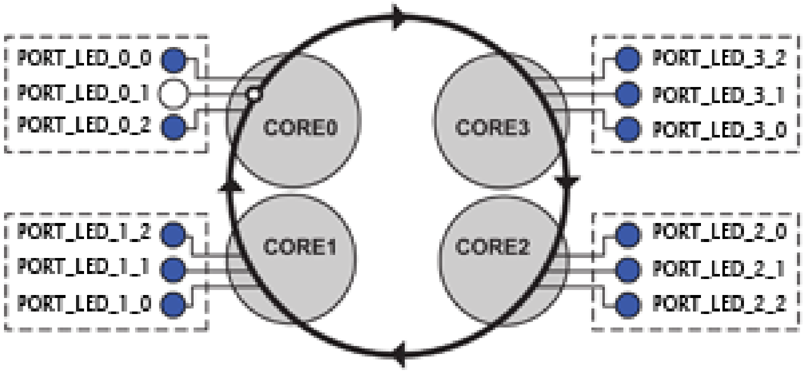
```
#include <platform.h>
#define PERIOD 20000000

out port cled0 = PORT_CLOCKLED_0;
out port cled1 = PORT_CLOCKLED_1;
out port cled2 = PORT_CLOCKLED_2;
out port cled3 = PORT_CLOCKLED_3;
out port cledG = PORT_CLOCKLED_SELG;
out port cledR = PORT_CLOCKLED_SELR;

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 neighbor */
  }
}

int main (void) {
  chan c0, c1, c2, c3;
  par {
    on stdcore [0]: { cledG <: 1;
                      tokenFlash (c0, c1, cled0, PERIOD, 1);
                    }
    on stdcore [1]: tokenFlash (c1, c2, cled1, PERIOD, 0);
    // other cores
  }
  return 0;
}
```

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

6.2 Examine the code

An XC channel provides a synchronous, bidirectional link between two threads. It consists of two channel ends, which two threads can use to interact on demand using the XC input and output statements.

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



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 `main` function constructs the token ring. 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 four 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 1

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

1. Modify the function tokenFlash so that it flashes each of the three LEDs connected to its port in sequence, add the required port declarations and complete the definition of main.
2. Build your application, create a new Run Configuration, and run it.
3. On your XC-1A, verify that the 12 LEDs each flash in sequence as the token cycles around the four threads, and then click the **Terminate** button () to stop your application running.

7 Use a button to change the LED color

This part of the tutorial shows you how to detect a button press and respond to it by changing the color of the LED cycling around the clockface, using the XC `select` statement.

7.1 Examine the code

A `select` statement is used to respond to one of a set of inputs, depending on which becomes ready first. If more than one 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.

```
void buttonListener ( chanend left ,
                      chanend right ,
                      in port button ,
                      out port g,
                      out port r) {
  int token ;
  int isGreen = 1;
  g <: 1;
  while (1)
    select {
      case left :> token :
      /* pass token on */
        right <: token ;
      break ;
      case button when pinsneq (0xf):> void :
        /* change color *
        ...
      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.

7.2 Exercise 1

To finish this part of the tutorial, complete the following tasks which build on the program from the previous section:

1. Add a declaration for the button port (see the XC-1A Hardware Manual[3] for details of the initializer).
2. Complete the function buttonListener and integrate it into the token ring.

   Note that you cannot output to a port in two concurrent threads, nor can you write the same variable in parallel. These restrictions prevent common programming errors such as race conditions, and ensure that the two threads can be run on any two cores, regardless of whether they share memory.
3. Run your application.

   Press one of the buttons to change the colour of an LED as it circulates around the clock.
4. On your XC-1A, verify that pressing a button changes the colour of an LED as it circulates, and then click the **Terminate** button () to stop your application running.

8 What to read next

This tutorial provides only a basic introduction the XC-1A hardware.

For more information on the board refer to the XC-1A Hardware Manual[4].

[3] http://www.xmos.com/published/xc1ahw
[4] http://www.xmos.com/published/xc1ahw

For more information on programming in XC see Programming XC on XMOS Devices[5].

[5] http://www.xmos.com/published/xc_en