

XC-1A Development Kit Tutorial

Version 9.9



Publication Date: 2010/02/01

Copyright © 2010 XMOS Ltd. All Rights Reserved.

1 Introduction

The XC-1A is a development board based on the XMOS XS1-G4 device. It comprises a single XS1-G4, 16 LEDs, four press-buttons, a speaker, JTAG and serial interfaces, and a through-hole prototyping area for connecting external components.

The XS1-G4 consists of four XCores, each comprising an event-driven processor with tightly integrated general purpose I/O. Each processor provides up to eight threads, 400 MIPS and 64 KBytes of RAM. The I/O pins are connected to board components and can be directly controlled by software (see Section 9).

Programs are written using a combination of XC, C and C++. XC provides extensions to C that simplify the control over concurrency, I/O and time. These extensions map directly onto XCore hardware resources such as threads and ports, and are *efficient*—compiling into short instruction sequences, and *safe*—free from many sources of deadlock, race conditions and memory violations. This makes programs easy to write, understand and debug.

This tutorial provides an introduction to writing XC programs using the components integrated on your XC-1A. It assumes that you are familiar with C [1]. This tutorial shows you 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 LED colour

The example programs show you how XC simplifies the creation of programs for event-driven processors. Sections of the tutorial that introduce a new language feature include the new keywords or operators in their title.

The example programs have been tested with version 9.9 of the Tools. Information on downloading, installing and using these tools is provided in a separate user guide [2].

2 Illuminate an LED: port, <:

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

A port connects a processor to one or more physical pins and is used to interface with the board components. The port logic can drive its pins high or low, or it can sample the value on its pins.

The program below illuminates a single LED on your XC-1A:

```
#include <platform.h>

out port bled = PORT_BUTTONLED;

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

The declaration

```
out port bled = PORT_BUTTONLED;
```



declares an output port named `bled`, which refers to the 4-bit port identifier `PORT_BUTTONLED` (see Section 9). This identifier is defined in the board description file “XC-1A.XN” and is exported to the header file `platform.h` during compilation.

Integrated input and output statements make it easy to express I/O operations on ports. In `main`, the statement

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

outputs the value `0b0001` to the port `bled`, causing the port to drive one of its four pins high. This in turn causes the LED connected to the pin to illuminate.

The port continues to drive the pins until instructed otherwise or until the program terminates. The infinite loop that follows the output statement prevents the latter condition.

-  Compile and run this program on your XC-1A. The LED labelled A should illuminate.
-  Modify the value output to the port so that all four button-LEDs are illuminated.

3 Flash an LED: timer, :>

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

A timer is a hardware resource 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 program below flashes an LED on your XC-1A:

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

out port bled = PORT_BUTTON_LED;

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;
}
```

The statement

```
tmr :> t;
```

inputs the value of the timer `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 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`.



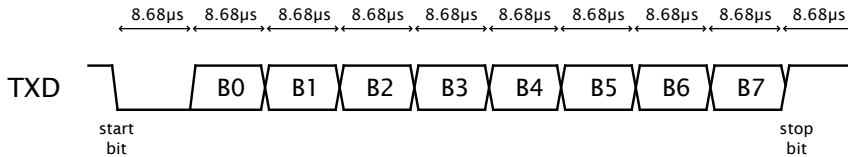
Compile and run this program on your XC-1A. The LED labelled A should flash at the rate defined by the flash period.

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.

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 $\frac{1}{115200} = 8.68\mu\text{s}$.

The program below serialises 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;
}
```

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 performing the shift as a separate operation afterwards.



The output-shift-right operator `<: >>` requires the output expression to be a variable of type `int`.

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. 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 above 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 diagram at the end of this tutorial on page 12. Compile and run this program on your XC-1A. The terminal should receive and display the message.

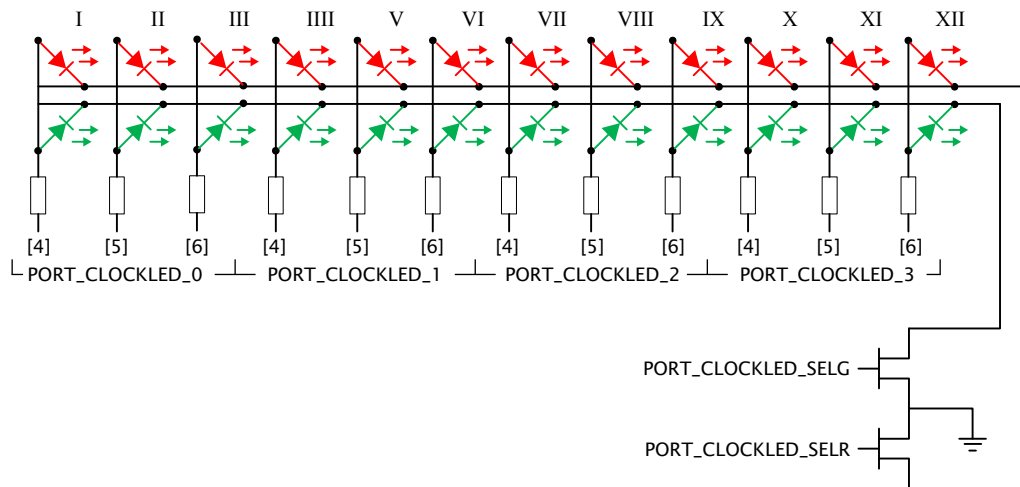
5 Flash multiple LEDs in parallel: `par`, `on`

This part of the tutorial shows you how to use the XC `par` and `on` statements to flash multiple clock-LEDs in parallel on your XC-1A.

The schematic for the 12 clock-LEDs is shown below; the location of the ports are shown in the diagram on page 12. 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.

¹Currently on MACs, the virtual COM port cannot be supported at the same time as the JTAG interface, preventing you from completing the following exercise.

²http://www.xmoslinkers.org/tag_search?tag=uart



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

```
#include <platform.h>
#define PERIOD 2000000

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

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.



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.



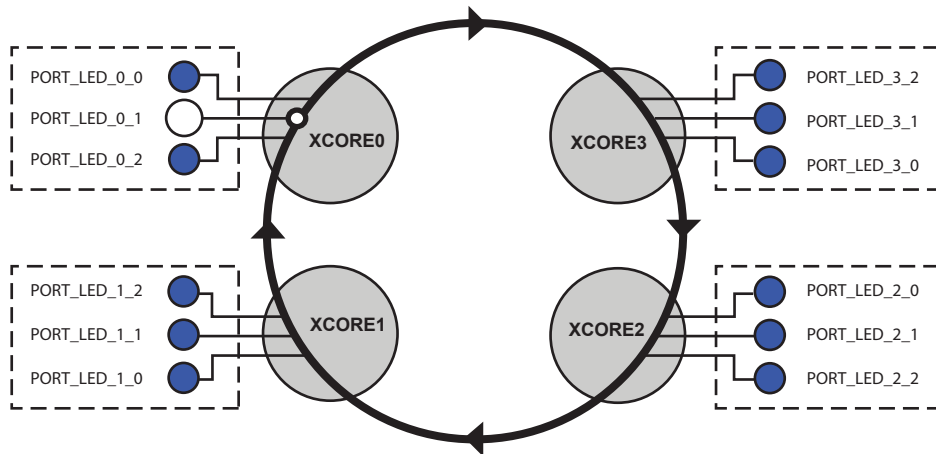
Implement the body of the function `flashLED` so that it flashes a single LED. Note that the pins are connected to bits 4–6 on the 8-bit port. Compile and run this program on your XC-1A. Four LEDs on the clockface should continually flash at a fixed rate.



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 Flash multiple LEDs in sequence: `chan`, `chanend`

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 illustrated below:



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 below implements a component of the token ring illustrated above, repeatedly inputting a token from its left neighbour, flashing an LED and outputting the token to its right neighbour:

```

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 neighbour */
        led <: 1;
        tmr :> t;
        tmr when timerafter(t+delay) :> void;
        led <: 0;
        right <: token; /* output token to right neighbour */
    }
}

```

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, c3;
    par {
        on stdcore[0]: { cledG <: 1;
                        tokenFlash(c0, c1, cled0, PERIOD, 1);
                      }
        on stdcore[1]: tokenFlash(c1, c2, cled1, 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 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`.



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`. Compile and run this program on your XC-1A. As the token cycles around the four threads, the 12 LEDs should each flash in sequence.

7 Add a button controller to the token ring: `select`

This part of the tutorial shows you how to detect a button press and respond by changing the colour of the LED cycling around the clockface, using the XC `select` statement. The example builds upon the token ring network developed in the previous section.

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 then 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 colour */
                ...
                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.

 Add a declaration for the button port (obtaining the initialiser from the diagram on page 12), complete the function `buttonListener` and integrate it into the token ring. Compile and run this program on your XC-1A. Pressing one of the buttons should cause the LED to change colour.

 Note that you cannot output to a port in two concurrent threads, nor can you write the same variable in parallel. These restrictions prevents 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.

 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 neighbour, and extend `buttonListener` similarly.

Congratulations, you're now ready to start developing programs using XC and your XC-1A board.

8 What to Read Next

This tutorial provides only a basic introduction the XC language and XMOS architecture. The following documents provide more information on designing with XC on your XC-1A card:

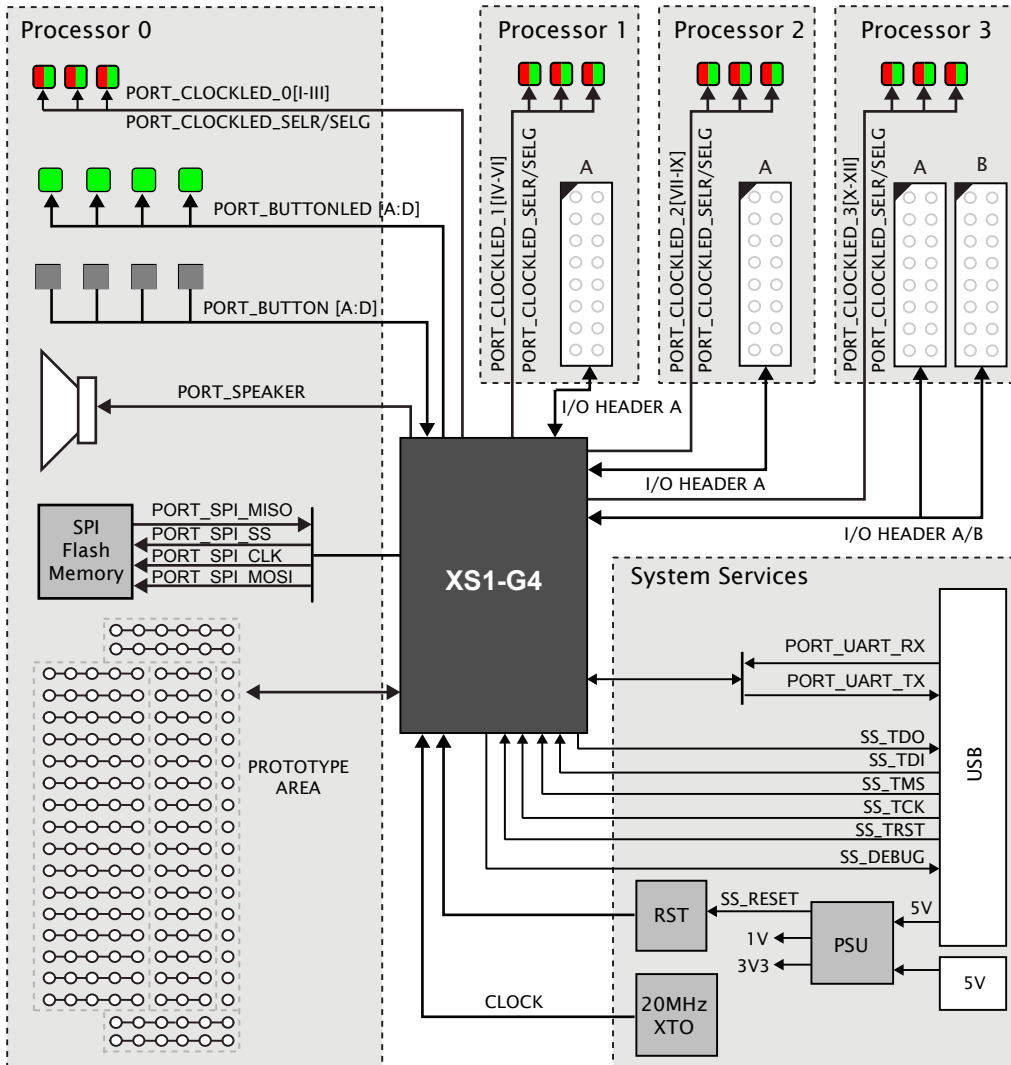
- **Programming XC on XMOS Devices** [3]: Provides an in-depth tutorial on how to write XC programs, including case studies of real-world designs, along with the official XC language specification and details of its implementation on XS1 devices.
- **XMOS Tools User Guide** [2]: Explains how to use the XMOS tools.
- **XC-1A Hardware Manual** [4]: Provides a functional description of the XC-1A board including the port-to-pin mappings of the through-hole prototyping areas, required to add additional components to your XC-1A based designs.

You may also find information from the following online resources useful:

- <http://www.xmos.com/>
- <http://www.xmoslinkers.org/>

9 XC-1A Port-To-Component Map

The diagram below shows how components on your XC-1A are connected to the processors on the XS1-G4, and gives the port names used in software.



Bibliography

- [1] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall Press, Upper Saddle River, NJ, USA, 1988.
- [2] Douglas Watt and Huw Geddes. *The XMOS Tools User Guide*. XMOS Limited, 2009. http://www.xmos.com/published/xtools_en.
- [3] Douglas Watt. *Programming XC on XMOS Devices*. XMOS Limited, Sep 2009. http://www.xmos.com/published/xc_en.
- [4] XMOS Ltd. XC-1A Hardware Manual. Website, 2010. <http://www.xmos.com/published/xc1ahw>.

Disclaimer

XMOS Ltd. is the owner or licensee of this design, code, or Information (collectively, the “Information”) and is providing it to you “AS IS” with no warranty of any kind, express or implied and shall have no liability in relation to its use. XMOS Ltd. makes no representation that the Information, or any particular implementation thereof, is or will be free from any claims of infringement and again, shall have no liability in relation to any such claims.

Copyright ©2009 XMOS Ltd. All Rights Reserved. XMOS and the XMOS logo are registered trademarks of XMOS Ltd in the United Kingdom and other countries, and may not be used without written permission. Company and product names mentioned in this document are the trademarks or registered trademarks of their respective owners. Where those designations appear in this document, and XMOS was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.