Application Note: AN00152

xSCOPE - Bi-Directional Endpoint

This application note shows how to create a simple example which uses the XMOS xSCOPE application trace system to provide bi-directional communication with a custom application running on a host machine.

The code associated with this application note demonstrates a simple console application running on a host PC which can communicate to the xCORE multicore microcontroller via the xSCOPE system.

The xTIMEcomposer development tools provide an xSCOPE endpoint library which can be used to interface a custom application into the xSCOPE server provided. This allows communication to and from the xCORE processor via a simple API and socket connection which can be enabled.

Example code for both the xCORE and host system is provided to enable an end-to-end demonstration of this capability.

Required tools and libraries

- xTIMEcomposer Tools - Version 13.2

Required hardware

This application note is designed to run on any XMOS xCORE multicore microcontroller

The example code provided with the application has been implemented and tested on the XMOS startKIT but there is no dependancy on this board and it can be modified to run on any development board which has xSCOPE support available. It can also be run on the XMOS simulator if required.

Prerequisites

- This document assumes familiarity with the XMOS xCORE architecture, 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].
- The XMOS tools manual contains information regarding the use of xSCOPE and how to use it via code running on an xCORE processor[2].

[1] http://www.xmos.com/published/glossary
[2] http://www.xmos.com/published/xtimecomposer-user-guide

1 Overview

1.1 Introduction

Debugging in circuit can be challenging. Once a system is subject to real-time stimuli it can be difficult to track down causes of unexpected behavior. Single stepping using a debugger may have little value since hard real-time systems break once a deadline has been missed.

This means debugging in circuit requires a way of monitoring the system and exporting data without affecting the code you are observing. The answer is low intrusive data collection, which must allow the system to run without changing its behavior or introducing timing effects. It is also desirable to observe high level and relevant data, such as state variables, input/output values in control loops or even directly observe data streams. This is exactly what xSCOPE provides; real-time, in-circuit instrumentation of user specified data probes, without affecting your design or device operation.

xSCOPE provides high level information at very high performance. Data rates of 1MSPS are possible allowing the multiple variables from inner control loops to be captured or high bit rate audio streams to be analyzed. The secret to this performance is a high speed USB 2.0 connection to the 4-wire xCONNECT port using the xTAG debug adapter. The xTAG debug adapter itself is powered by an xCORE multicore microcontroller.

The XMOS xTIMEcomposer tools provide an interface to allow 3rd party and user applications to connect into the xSCOPE server which runs on the XMOS xTAG. This allows custom applications to be built which can process the data being generated by the xCORE multicore microcontroller. In addition to output the system also allows data to be uploaded back to the target device to allow more complex feedback systems to be built.

1.2 Block diagram

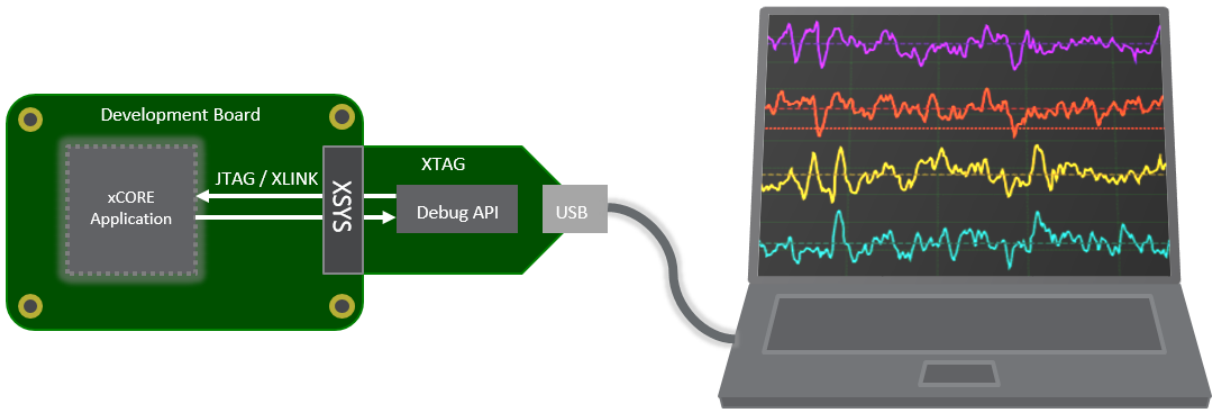

Figure 1: Block diagram of xSCOPE bi-directional endpoint example

2 xSCOPE bi-directional endpoint example

The example in this document does not have any external dependancies on application libraries other than those supplied with the XMOS development tools. It demonstrates how to interface code running on an xCORE tile with an application running on a host machine using xSCOPE. This simple example shows the additions that are required to XMOS makefiles to enable xSCOPE and also how to use the xSCOPE host endpoint library to build a custom endpoint application.

This example is implemented using one task which responds to events coming into the xCORE tile from the xTAG development adapter via the xCONNECT network. This task deals with xSCOPE data arriving from the xTAG, processing this data and then responding.

The following diagram shows the task and communication structure for this bi-directional xSCOPE endpoint example.

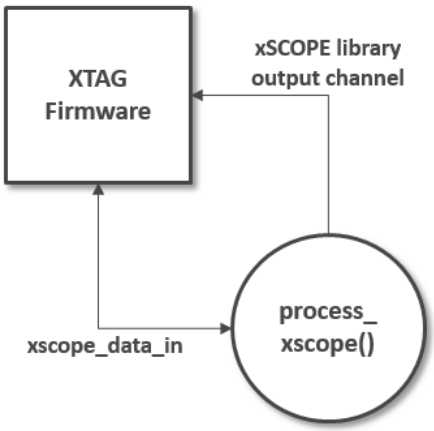

Figure 2: Task diagram of xSCOPE bi-directional endpoint example

2.1 Makefile additions for this application

It is simple to enable xSCOPE in an xCORE application. To link the xSCOPE library and perform the configuration required by an application the -fxscope option needs to be passed to the xCORE build tools in the makefile:

```
XCC_FLAGS = -Wall -O2 -report -fxscope
```

2.2 Setting up the xSCOPE configuration file

To configure the application to use xSCOPE a configuration file is required to set up the system. In this application the xSCOPE configuration file sets the basic I/O redirection mode so that Host application sends print messages to the embedded application via the xSCOPE link. The print messages are then transmitted back to our host application and printed from its console.

The xSCOPE configuration file for this example is as follows and is contained within the src directory of the xCORE application code.

```
<xSCOPEconfig ioMode="basic" enabled="true"> </xSCOPEconfig>
```

2.3 Declaring resource and setting up the xCORE tile

The example code in this application note is contained in a single file which contains a simple main() function and a function to handle the data being transmitted from the host application via xSCOPE. This application uses xCONNECT channels to transfer xSCOPE data to and from a host machine via the XMOS xTAG development adapter. There is only one task running in this application example on the xCORE tile.

2.4 Using xSCOPE on the xCORE tile

The xSCOPE library functions used to communicate with the xSCOPE server on the xTAG are accessed using the following header file:

```
#include <xscope.h>
```

2.5 The xCORE application main() function

The main() function for the xCORE tile is contained within the file `main.xc` and is as follows,

```
int main (void) {
  chan xscope_data_in;

  par {
    xscope_host_data(xscope_data_in);
    on tile[0]: process_xscope(xscope_data_in);
  }

  return 0;
}
```

Looking at this function in more detail you can see the following:

- There are two tasks defined in the par statement inside main
- The function xscope_host_data() is a service which connects the xCORE tile to the xTAG via an xCONNECT channel
- The xscope_host_data() function does not execute at runtime, it is used by the build tools to constuct the connection between the xTAG and a local xCONNECT channel end
- The function process_xscope() is used to handle data coming into the xCORE tile via the xSCOPE system
- The function process_xscope() takes a channel end which is connected to the xTAG as an argument

2.6 The xCORE application process_xscope() function

```
void process_xscope(chanend xscope_data_in) {
  int bytesRead = 0;
  unsigned char buffer[256];

  xscope_connect_data_from_host(xscope_data_in);

  while (1) {
    select {
      case xscope_data_from_host(xscope_data_in, buffer, bytesRead):
      if (bytesRead) {
        printstr(buffer);
        if (buffer[0] == 'q')
          return;
      }
      break;
    }
  }
}
```

Looking at this function in more detail you can see the following:

- The function takes a channel end as input which is connected to the xTAG
- There are variables defined for a byte count and a byte buffer
- The function xscope_connect_data_from_host() is used to configure and synchronize the channel end xscope_data_in into bi-directional mode xTAG
- There is a while loop which executes until a quit message is received from the host
- The function xscope_data_from_host() is used within a select statement to wait for a data available event signalling that xSCOPE data is available
- The bytesRead variable contains the number of bytes available to process
- The buffer received is sent back to the host via the printstr() fuction which is redirected via xSCOPE and will be sent to the host application
- If a q character is received from the host the while loop will exit

2.7 The host application main() function

The main() function for the host application is contained within the file `main.cpp` and is as follows,

```
int main (void) {
  char data[1024];

  xscope_ep_set_print_cb(xscope_print);
  xscope_ep_connect("localhost", "10234");

  printf("----- xSCOPE Console Demo -----\n");

  while(data[0] != 'q') {
    if (fgets(data, 1024, stdin)) {
      xscope_ep_request_upload(strlen(data)+1, (const unsigned char *)data);
    }
  }
  return 0;
}
```

Looking at this function in more detail you can see the following:

- A data buffer is declared to transmit data to the xCORE tile via xSCOPE
- The function xscope_ep_set_print_cb() is used to register a call back which handles print messages from the xCORE tile
- The function xscope_ep_connect() is used to initialize a connection to the xSCOPE server running within the XMOS development tools
- The application uses the function fgets() to read a string from standard input
- Data is transmitted from the host to the xCORE tile using the xscope_ep_request_upload() function
- The main() function will terminate when the user enters q character on the console

2.8 The host application xscope_print() function

The xscope_print() function in this application is a callback which is registered which will be called when the xSCOPE endpoint receives an output message from the xCORE tile which is via one of the standard xCORE tile printing routines such as printstr(). The code for this function is as follows,

```
void xscope_print(unsigned long long timestamp,
                  unsigned int length,
                  unsigned char *data) {
  if (length) {
    printf("--> ");
    for (int i = 0; i < length; i++) {
      printf("%c", *(&data[i]));
    }
  }
}
```

This function performs the following operation:

- The message timestamp (if available) is passed as an argument to the function
- The message length is passed as an argument to the function
- The message data is passed as an argument to the function
- The function loops over the data received and outputs the characters to standard output

APPENDIX A - Example Hardware Setup

The application example is designed to run on an XMOS startKIT. As discussed earlier the xCORE tile code can be run on any development board supporting xSCOPE by changing the board target in the XMOS Makefile.

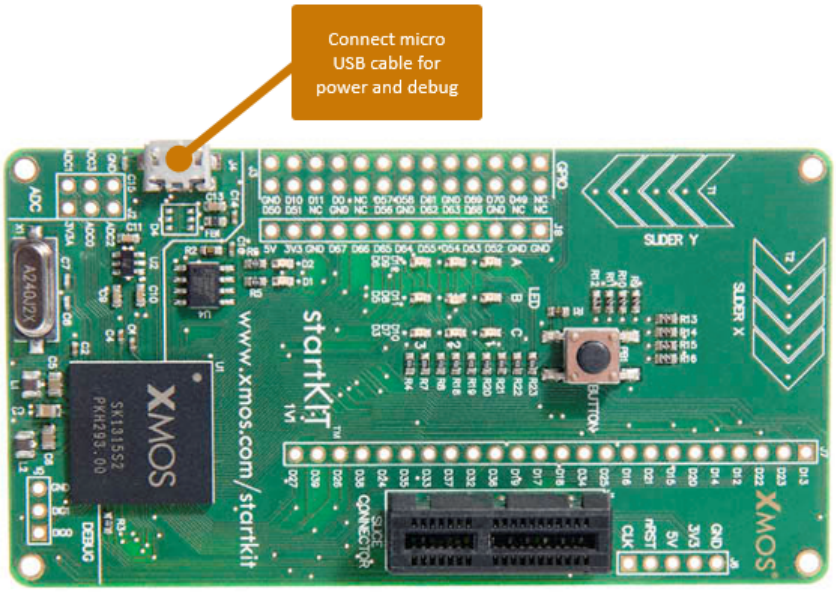

Figure 3: XMOS startKIT

The hardware should be configured as displayed above for this demo:

- A micro USB cable should be connected between the XMOS startKIT and the host machine to provide power and access to the xTAG for the XMOS development tools

APPENDIX B - Host Application Compilation

The host application which provides the console input and output is provided in both source and binary form on the platforms supported by the XMOS development tools. To recompile the host application you will need to have a working compiler for your host environment. Instructions to rebuild the host application for each platform are below.

These commands are executed from a command prompt at the top level of the application note directory. The environment should be set up with the XMOS tools on your path by using the script to setup the tools provided in the tools installation directory

Win32:

```
cl /o xscope_host_console.exe src\host\main.cpp -I"%XMOS_TOOL_PATH%\include" "%XMOS_TOOL_PATH%\lib\
  ↪ xscope_endpoint.lib"
```

OSX:

```
gcc -o xscope_host_console src/host/main.cpp -I $XMOS_TOOL_PATH/include $XMOS_TOOL_PATH/lib/xscope_endpoint.so
```

Linux32:

```
gcc -o xscope_host_console src/host/main.cpp -I $XMOS_TOOL_PATH/include $XMOS_TOOL_PATH/lib/xscope_endpoint.so
```

Linux64:

```
gcc -o xscope_host_console src/host/main.cpp -I $XMOS_TOOL_PATH/include $XMOS_TOOL_PATH/lib/xscope_endpoint.so
```

APPENDIX C - Launching the demo application

C.1 Launching from the command line

From the command line, the xrun tool is used to download code to the xCORE-USB device. To enable the xSCOPE server to allow our host application to connect you need to pass the specfic xSCOPE command line option.

Changing into the bin directory of the project, the code can be executed on the xCORE microcontroller as follows:

```
> xrun --xscope-port localhost:10234 app_xscope_console.xe          <-- Download and execute the xCORE code
```

At this point the xrun command will be started but waiting for a connection from the host application on port 10234 of the local machine.

Now launch the host application to continue.

C.2 Launching from xTIMEcomposer Studio

From xTIMEcomposer Studio a run configuration is used to download code to the xCORE device. Select the xCORE binary from the bin directory, right click and then follow these steps: -0.5em

1. Select **Run As**.
2. Select **Run Configurations**.
3. Double click on xCORE application**.
4. Enable xSCOPE in Target I/O options:

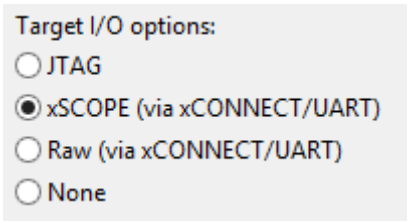

Figure 4: xTIMEcomposer xSCOPE configuration

5. Add the additional command line option –xscope-port localhost:10234

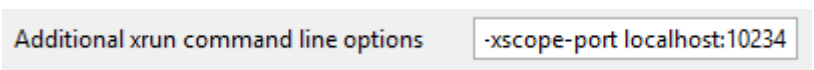

Figure 5: xTIMEcomposer additional xrun options

6. Click **Apply** and then **Run**.

Now launch the host application to continue.

APPENDIX D - Launching the xSCOPE endpoint host application

If you have recompiled the host application, you can run that from the location where this has been built.

Precompiled binaries for the xSCOPE endpoint have been provided in the top level host directory for the target platforms.

To run the endpoint change directory into the location for your host platform and execute the xscope_host_console binary from your console.

For example on Windows this should work as follows:

```
>xscope_host_console.exe
----- xSCOPE Console Demo -----
  > hello
  --> hello
  > This is a test
  --> This is a test
  > q
```

Once the q character is passed to the host application both the xCORE tile and the host xSCOPE endpoint will exit.

APPENDIX E - 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 F - Full source code listing

F.1 xCORE source code for main.xc

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#include <platform.h>
#include <xscope.h>
#include <print.h>

void process_xscope(chanend xscope_data_in) {
  int bytesRead = 0;
  unsigned char buffer[256];

  xscope_connect_data_from_host(xscope_data_in);

  while (1) {
    select {
      case xscope_data_from_host(xscope_data_in, buffer, bytesRead):
      if (bytesRead) {
        printstr(buffer);
        if (buffer[0] == 'q')
          return;
      }
      break;
    }
  }
}

int main (void) {
  chan xscope_data_in;

  par {
    xscope_host_data(xscope_data_in);
    on tile[0]: process_xscope(xscope_data_in);
  }

  return 0;
}
```

F.2 Host source code for main.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <xscope_endpoint.h>

void xscope_print(unsigned long long timestamp,
                  unsigned int length,
                  unsigned char *data) {
  if (length) {
    printf("--> ");
    for (int i = 0; i < length; i++) {
      printf("%c", *(&data[i]));
    }
  }
}
```

```
int main (void) {
  char data[1024];

  xscope_ep_set_print_cb(xscope_print);
  xscope_ep_connect("localhost", "10234");

  printf("----- xSCOPE Console Demo -----\n");

  while(data[0] != 'q') {
    if (fgets(data, 1024, stdin)) {
      xscope_ep_request_upload(strlen(data)+1, (const unsigned char *)data);
    }
  }
  return 0;
}
```