Application Note: AN00151

xSCOPE - Custom Host Endpoint

This application note shows how to create a simple example which uses the XMOS xSCOPE application trace system to provide instrumentation logging to 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

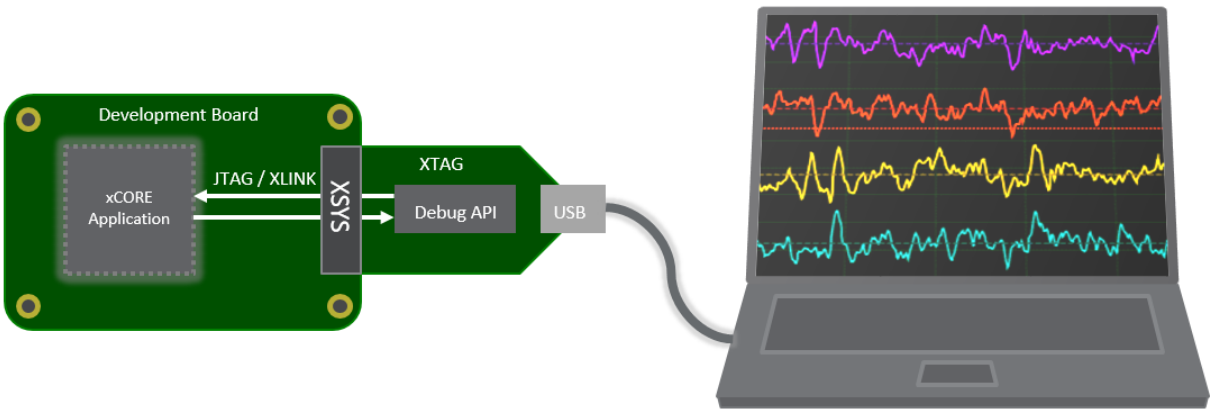

Figure 1: Block diagram of xSCOPE custom host endpoint

2 xSCOPE custom host endpoint example

The example in this document has no 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.

The following diagram shows the task and communication structure for this xSCOPE custom host endpoint example.

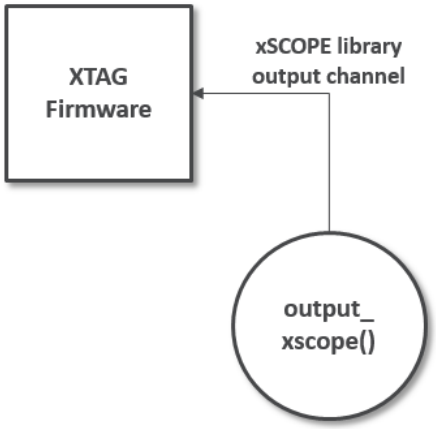

Figure 2: Task diagram of xSCOPE custom host endpoint example

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 simply sets the basic I/O redirection mode so that print messages from the xCORE tile are transmitted via xSCOPE rather than via JTAG. In this application the print messages are transmitted back to our host application and printed to its console.

In addition to print redirection this example defines a new custom xSCOPE probe which can be used from the xCORE tile to report state. These are also defined within the config.xscope file.

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">
  <Probe name="Value" type="CONTINUOUS" datatype="UINT" units="Value" enabled="true"/>
</xSCOPEconfig>
```

2.3 Declaring resource and setting up the xCORE tile

The example code in this application note is contained within a single file which contains an simple main() function and a function to handle the data being being output from the xCORE tile to the host application. The xSCOPE library is being used in this example to instrument embedded application code in order to report the status of variables within the host application.

2.4 Using xSCOPE on the xCORE tile

The xSCOPE library functions used to communicate data to the host machine 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) {
  par {
    on tile[0]: output_xscope();
  }

  return 0;
}
```

- There is a single task defined in the par statement inside main
- The function output_xscope() is used to capture instrumentation of the executing program via xSCOPE

2.6 The xCORE application output_xscope() function

```
void output_xscope() {
  printstr("Starting xSCOPE probe output");
  delay_milliseconds(1);
  for (int i = 0; i < 25; i++) {
    xscope_int(VALUE, i);
    delay_milliseconds(1);
  }
  printstr("Finished xSCOPE probe output");
  // Send q to signal host application exit
  printstr("q");
}
```

- The function outputs print messages which are redirected to the host application via xSCOPE
- The xscope_int() function is used to capture the state of the loop variable `i`
- A 1 millisecond delay is introduced using the delay_milliseconds() library function
- The q character is printed at the end of the function to cause the host application to terminate

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) {
  xscope_ep_set_print_cb(xscope_print);
  xscope_ep_set_register_cb(xscope_register);
  xscope_ep_set_record_cb(xscope_record);
  xscope_ep_connect("localhost", "10234");

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

  while(running);

  return 0;
}
```

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

- 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_set_register_cb() is used to register a call back which handles event registration messages from the xSCOPE server
- The function xscope_ep_set_record_cb() is used to register a call back which handles event record messages from the xSCOPE server
- The function xscope_ep_connect() is used to initialize a connection to the xSCOPE server running within the XMOS development tools
- The main() function will terminate when the global flag running is cleared by the application

2.8 The host application xscope_print() function

The xscope_print() callback 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) {
    if (data[0] == 'q') {
      running = 0;
      return;
   }

    for (int i = 0; i < length; i++)
      printf("%c", *(&data[i]));

    printf("\n");
  }
}
```

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
- When a q character is received from the xCORE tile via a print message the global flag running is set to 0 which causes the host application to exit

2.9 The host application xscope_register() function

The xscope_register() callback will be called when the xSCOPE server transmits event probe registration messages. These relate to the xSCOPE probes which are registered in the config.xscope file.

```
void xscope_register(unsigned int id,
                     unsigned int type,
                     unsigned int r,
                     unsigned int g,
                     unsigned int b,
                     unsigned char *name,
                     unsigned char *unit,
                     unsigned int data_type,
                     unsigned char *data_name) {

   printf("xSCOPE register event (id [%d] name [%s])\n",
          id, name);
}
```

This function performs the following operation:

- The record id is passed to the function
- The record type is passed to the function
- The record event colour is passed as RGB values to the function
- The record name is passed to the function
- The record time unit is passed to the function
- The record unit data type is passed to the function
- The record unit data type name is passed to the function
- The function prints out the record id and name for every registration event it receives

2.10 The host application xscope_record() function

The xscope_record() callback will be called when the xSCOPE server transmits a record event for a registered xSCOPE probe. This is the event information which is transmitted when an xSCOPE library function is called on the xCORE tile.

```
void xscope_record(unsigned int id,
                   unsigned long long timestamp,
                   unsigned int length,
                   unsigned long long dataval,
                   unsigned char *databytes) {
   float mstimestamp = timestamp / 1000000000.0f;

   printf("xSCOPE record event (id [%d] timestamp [%.3f ms] value [%lld])\n",
          id, mstimestamp, dataval);

}
```

This function performs the following operation:

- The record id is passed to the function
- The record timestamp is passed as an argument to the function
- The record length is passed as an argument to the function
- The record data value is passed as an argument to the function
- The record data bytes is passed to the function, this will be null for some xSCOPE event types
- The function prints the event id, timestamp converted to milliseconds and event value to the console.

APPENDIX A - Example Hardware Setup

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

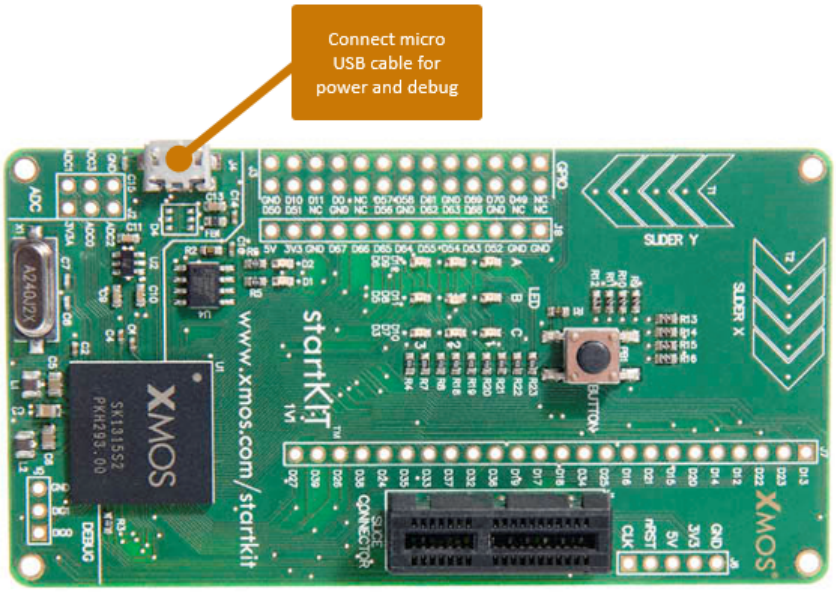

Figure 3: XMOS startKIT

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 compilation 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_endpoint.exe src\host\main.cpp -I"%XMOS_TOOL_PATH%\include" "%XMOS_TOOL_PATH%\lib\
  ↪ xscope_endpoint.lib"
```

OSX:

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

Linux32:

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

Linux64:

```
gcc -o xscope_host_endpoint 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 startKIT. To enable the xSCOPE server to allow our host application to connect you need to pass the specific 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_endpoint.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 the run mechanism 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:

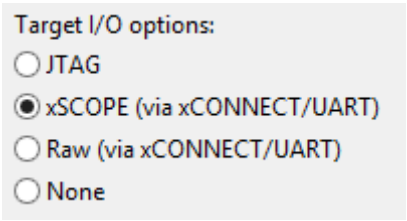

Figure 4: xTIMEcomposer xSCOPE configuration

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

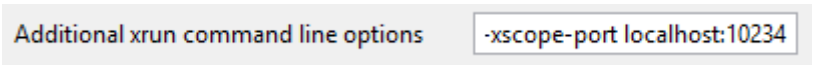

Figure 5: xTIMEcomposer additional xrun options

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

Now launch the host application to continue.

C.3 Launching the xSCOPE endpoint host application

If you have recompiled the host application then 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_endpoint binary from your console.

For example on Windows this should work as follows:

```
>xscope_host_endpoint.exe

----- xSCOPE Endpoint Demo -----
xSCOPE register event (id [0] name [Value])
Starting xSCOPE probe output
xSCOPE record event (id [0] timestamp [1.006 ms] value [0])
xSCOPE record event (id [0] timestamp [2.007 ms] value [1])
xSCOPE record event (id [0] timestamp [3.007 ms] value [2])
xSCOPE record event (id [0] timestamp [4.008 ms] value [3])
xSCOPE record event (id [0] timestamp [5.008 ms] value [4])
xSCOPE record event (id [0] timestamp [6.009 ms] value [5])
xSCOPE record event (id [0] timestamp [7.009 ms] value [6])
xSCOPE record event (id [0] timestamp [8.010 ms] value [7])
xSCOPE record event (id [0] timestamp [9.010 ms] value [8])
xSCOPE record event (id [0] timestamp [10.011 ms] value [9])
xSCOPE record event (id [0] timestamp [11.011 ms] value [10])
xSCOPE record event (id [0] timestamp [12.012 ms] value [11])
xSCOPE record event (id [0] timestamp [13.012 ms] value [12])
xSCOPE record event (id [0] timestamp [14.012 ms] value [13])
xSCOPE record event (id [0] timestamp [15.013 ms] value [14])
xSCOPE record event (id [0] timestamp [16.013 ms] value [15])
xSCOPE record event (id [0] timestamp [17.014 ms] value [16])
xSCOPE record event (id [0] timestamp [18.014 ms] value [17])
xSCOPE record event (id [0] timestamp [19.015 ms] value [18])
xSCOPE record event (id [0] timestamp [20.015 ms] value [19])
xSCOPE record event (id [0] timestamp [21.016 ms] value [20])
xSCOPE record event (id [0] timestamp [22.016 ms] value [21])
xSCOPE record event (id [0] timestamp [23.017 ms] value [22])
xSCOPE record event (id [0] timestamp [24.017 ms] value [23])
xSCOPE record event (id [0] timestamp [25.018 ms] value [24])
Finished xSCOPE probe output
```

APPENDIX D - 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 E - Full source code listing

E.1 xCORE source code for main.xc

```
// Copyright (c) 2015, XMOS Ltd, All rights reserved
#include <xs1.h>
#include <platform.h>
#include <xscope.h>
#include <print.h>

void output_xscope() {
  printstr("Starting xSCOPE probe output");
  delay_milliseconds(1);
  for (int i = 0; i < 25; i++) {
    xscope_int(VALUE, i);
    delay_milliseconds(1);
  }
  printstr("Finished xSCOPE probe output");
  // Send q to signal host application exit
  printstr("q");
}

int main (void) {
  par {
    on tile[0]: output_xscope();
  }

  return 0;
}
```

E.2 Host source code for main.cpp

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

static volatile unsigned int running = 1;

void xscope_print(unsigned long long timestamp,
                  unsigned int length,
                  unsigned char *data) {
  if (length) {
    if (data[0] == 'q') {
      running = 0;
      return;
   }

    for (int i = 0; i < length; i++)
      printf("%c", *(&data[i]));

    printf("\n");
  }
}

void xscope_register(unsigned int id,
```

```
                      unsigned int type,
                      unsigned int r,
                      unsigned int g,
                      unsigned int b,
                      unsigned char *name,
                      unsigned char *unit,
                      unsigned int data_type,
                      unsigned char *data_name) {

   printf("xSCOPE register event (id [%d] name [%s])\n",
          id, name);
}

void xscope_record(unsigned int id,
                   unsigned long long timestamp,
                   unsigned int length,
                   unsigned long long dataval,
                   unsigned char *databytes) {
   float mstimestamp = timestamp / 1000000000.0f;

   printf("xSCOPE record event (id [%d] timestamp [%.3f ms] value [%lld])\n",
          id, mstimestamp, dataval);

}

int main (void) {
  xscope_ep_set_print_cb(xscope_print);
  xscope_ep_set_register_cb(xscope_register);
  xscope_ep_set_record_cb(xscope_record);
  xscope_ep_connect("localhost", "10234");

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

  while(running);

  return 0;
}
```