Application Note: AN01024

xCONNECT dynamic configuration demo

xCONNECT is a proprietary interconnect technology that facilitates data communication across different xCORE to create a fully scalable system. It is possible to achieve high bandwidth communication of up to 400 Mbits/sec for each xCONNECT link making it suitable for things like light weight industrial back-plane busses. No additional hardware is required for the xCONNECT communication.

Using xCONNECT over longer distances can introduce bit errors due to noise, xCONNECT is dependent on the application layer to recover from such communication errors. This application note demonstrates handling of transmit timeouts, receive timeouts and receive exceptions (e.g. unexpected control tokens) using software to ensure robustness of the communication.

Required tools and libraries

- xTIMEcomposer Tools - Version 14.0.0 and above
- XMOS try_catch exception handling module - Version 1.0.5 and above

Required hardware

This application note is designed to run on an XMOS xCORE General Purpose (L-series) device.

The example code provided with the application has been implemented and tested on the xCORE L-series sliceKIT core board 1V2 (XP-SKC-L2) but there is no dependency on this board and it can be modified to run on any development board which uses an xCORE General Purpose (L-series), xCORE-USB series or xCORE-Analog series device.

Prerequisites

- This document assumes familiarity with the XMOS xCORE architecture, xCONNECT interconnect communication, the XMOS tool chain and the xC language. Documentation that is not specific to this application note is listed in the references appendix.
- For descriptions of XMOS related terms found in this document please see the XMOS Glossary[1].

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

1 Overview

1.1 Introduction

Each xCORE device has an xCONNECT infrastructure to provide scalable interconnect between cores and tiles. A transfer rate of up to 400Mbits/sec is achievable over a given xCONNECT link using a suitable communication mode.Such a communication link is useful in applications such as

- industrial back-plane communication where the protocol requires a simple and light weight link and network layer handling unlike protocols such as EtherCAT, PROFINET
- off chip communication extenders between xCORE devices, xCORE and FPGA based host peripherals etc.

Refer to xCONNECT architecture and XS1-L system specification for more information.

1.2 Block diagram

xCONNECT consists of links and switches to transfer packets across xCORE tiles:

- cores communicate over channels
- Plinks provide intra-tile communication as well as connection between channel ends and device switch (called sswitch)
- xCONNECT (XMOS links) provide inter-tile and off-chip communication.

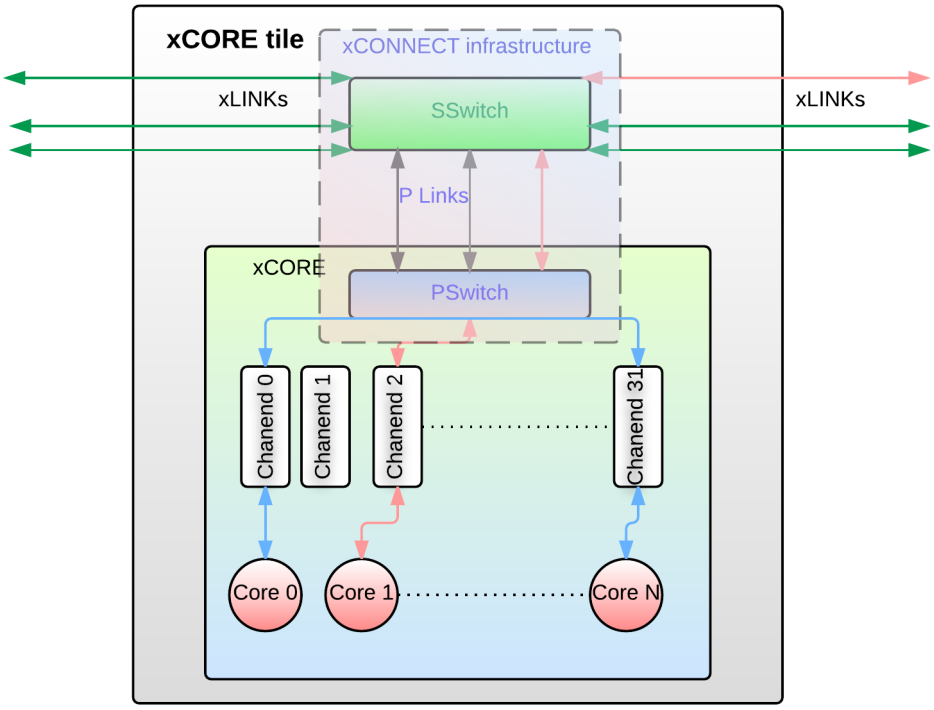

Figure 1: Block diagram of an xCORE tile

2 xCONNECT demo application

This application demonstrates the following features:

- Handle xCONNECT token errors, preventing exception caused by an invalid token and recovering
- Implement plug and unplug detection of the device in the communication bus using non-routed links
- Handle the transmit and receive communication timeout so that the task is not blocked if the data is not consumed on the other side of the channel

In order to ensure data consistency, error correction and control may be added to the software. Refer to FAQs section for more details on how this might be achieved.

The demo code is split into two application images. Each of them boots from JTAG independently, at different start up times. An application (called the transmit application) sends a continuous stream of data to another xCORE tile using xCONNECT links. Another application (called the receive application executing on another hardware board) receives this xCONNECT data using a channel.

An application (called a transmit application) sends a continuous stream of data to another xCORE tile using xCONNECT links. Another application (called a receive application available on a different xCORE tile) receives this xCONNECT data using a channel. A reporting task uses this data to report the communication parameters like the link bandwidth, token errors, and number of timeouts.

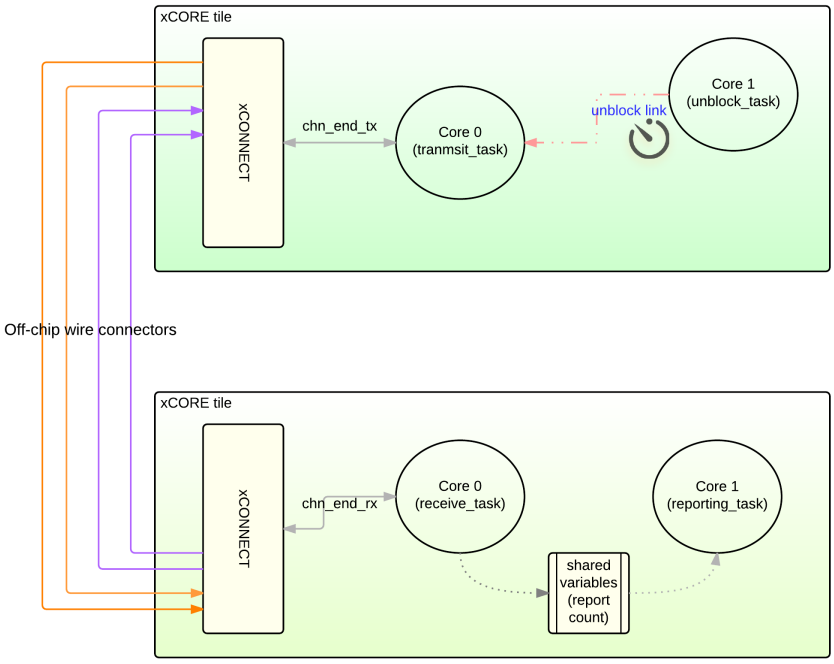

Figure 2: Demo application tasks

2.1 The transmit application main() function

Below is the source code for the main function of this application, which is taken from the source file `main.xc`

```
int main(void)
{
  par {
    on tile[0] : transmit_link();
    on tile[0] : re_enable_tx_link();
  }
  return 0;
}
```

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

- The par statement describes running two separate tasks in parallel
- transmit_link task attaches an exception handler and invokes *transmit_handler*
- transmit_handler is the application task which operates in while (1), sends data tokens over the xCONNECT and maintains the transmit state of the task. Its functionality is explained in the below sections
- re_enable_tx_link operates in a loop and periodically unblock the xLINK used by transmit_handler

2.2 Configuring links in non-routed static mode

A bi-directional XMOS link needs to be setup for the physical and link layer communication. xCONNECT assumes that the signal is correct. Any fake edge will cause extra bits to appear. Any such extra edge will cause extra bits and the END-OF-MESSAGE is destroyed, or randomly introduced causing header corruption. It will then be randomly routed resulting in communication errors.

The best way to use an xCONNECT in such cases is to avoid routing and perform error detection using a software protocol. In the statically routed links, no header is required and routing is not attempted. This setup requires node register and link direction configured manually (in the application code).

```
case 0: //setup link direction
  int reg_val = 0;
  int direction = 0x1;
  reg_val = XS1_LINK_DIRECTION_SET(reg_val, direction);
  write_node_config_reg(tile[0], XS1_SSWITCH_SLINK_0_NUM + LINK_NUM, reg_val);
  comm_state = 1;
```

Once this link and the direction is setup, the task sends the data using a channel end resource. In the code below, *c* is delcared *unsafe*; gets a channel end resource and it is passed to chan_setd(). The traffic on the link is then configured to deliver to a fixed destination channel end. Setup the destination node_id, its channel resource number and its type in the correct format as mentioned in the register setting (Node=0x8003, chanend number=00, resource type=02(chanend))

```
case 1: //channel setup
  unsafe {
    c = chan_getr();
    chan_setd(c, 0x80030002);
  }
  comm_state = 2;
```

Note that if the code is to be run on a different tile, modify the node_id according to the `receive application` tile id, as printed from the receive application's console.

Setup the chosen link to use static routing in the system switch configuration register.

```
int ret = write_sswitch_reg(get_local_tile_id(), XS1_L_SSWITCH_XSTATIC_0_NUM + LINK_NUM, 0x80000000ul);
delay_milliseconds(50);
comm_state = 4;
```

2.3 Hot-plug detection and handling

The link layer governs when the data is transmitted, and how the configured links start communicating. For hot-plug detection, xLINKs are initialised on the fly. The application layer sends transmit credit requests by sending *hello* tokens and waits to receive the *credit* token that is sent from the other side of the link.

```
do {
  link_reset(LINK_NUM);
  link_hello(LINK_NUM);
  delay_microseconds(100);
} while (!link_got_credit(LINK_NUM));
printf("Got credit\n");
err_ctr = 0;
comm_state = 5;
```

2.4 Transmit data

Once the credits are available, tx_handler routine streams out data tokens using the channels.

```
unsafe { c <: 'a'; }
```

Application task inserts periodic control tokens, which are useful for setting up known synchronisation points to recover from the communication errors.

```
if (err_ctr++ == SEND_CTRL_TOKEN) {
  err_ctr = 0;
  unsafe {
    outct((chanend)c, XS1_CT_ACK);  //Send a control token as app sync point
  }
}
```

2.5 Transmit timeout handling

Transmit task may be blocked due to a paused out instruction and the application is required to handle the transmit timeout. This is handled by performing disable, re-enable, and reset of the link. This example uses a separate *re_enable_tx_link* task to periodically re-enable the links as follows:

```
void re_enable_tx_link()
{
  while (1) {
    delay_seconds(RE_ENABLE_TX_PERIOD);
    SET_SHARED_GLOBAL(g_comm_state, 1);
    printf("Re-enable transmit link. Waiting for credit..\n");
    link_disable(LINK_NUM);
    link_enable(LINK_NUM);
  } //while (1)
```

Instead of using a separate core, timeout handling may use a timer interrupt to jump to an interrupt handler (setjmp/getjmp) to disable, re-enable and reset the link.

2.6 Exception handling

Trap handlers implemented in try-catch module perform a recovery needed for the application layer to proceed; nature of recovery depend on the type and place at which exception happened; in this example, recovery does re-enable and reset the link.

```
exception_t exception;

TRY {
  printf("Demo started...tile id: %x\n",get_local_tile_id());
  transmit_handler(0);
} CATCH (exception) {
  printf("Got exception.\n");
  transmit_handler(4);
```

2.7 The receive application main() function

Below is the source code for the main function of the receive application, which is taken from the source file `main.xc`

```
int main(void)
{
  par {
    on tile[DEMO_TILE] : receive_link();
    on tile[DEMO_TILE] : reporting_task();
  }
  return 0;
}
```

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

- The par statement describes running two separate tasks in parallel
- recieve_link task attaches an exception handler to rx_handler
- rx_handler is the application task which operates in while (1) resposnding to xCONNECT data events. Its functionality is dealt in more detail in the following sections
- reporting_task periodically wakes up, reads the total data bytes received and reports them in the console

Steps for configuring and setting up the link for the receive side application is the same as described in the transmit task sections above.

2.8 Receive data

The receiver continuously loops to collect the xCONNECT data using the channel end. This takes care of control token errors by handling the case where a control or data token is unexpected.

```
void test_ct(unsafe streaming chanend c, int &j)
{
  char r = 'z';
  unsafe {
    if (testct((chanend)c)) {
      r = inct((chanend) c);
      j++;
    }
    else {
      c :> r;
      unsafe {
        volatile int * unsafe p = &g_data_tokens;
        *p = *p + 1;
      }
    }
  }
```

Received tokens are updated in the global variables so that the reporting task can use them. In general, the receiver handles interspersing of data and user defined control tokens. This can be useful at the receiver to check for these user defined control tokens which act as a known synchronisation points.

2.9 Receive timeout handling

A timer (configured as timeout) event in the select handler breaks the task's continuous wait when the incoming data stream is not available. This event breaks the receive loop and resets it back to the credit exchange state as follows.

```
case rx_loop_tmr when timerafter(t + RX_TIME_OUT_TICKS) :> void:
  SET_SHARED_GLOBAL(g_ctrl_tokens, ctrl_tkn_ctr);
  SET_SHARED_GLOBAL(g_timeout_cnts, tm_out_ctr);
  printf("\nTimed out...\n\n");
  rx_loop = 0;
  comm_state = 3;
```

It waits for the credit from the other side before resuming further receive of the input stream.

APPENDIX A - Demo hardware setup

In order to set up 2-wire xCONNECT connection between two sliceKIT boards using direct cable wires, select two *xLINK B* pins and connect them as follows:

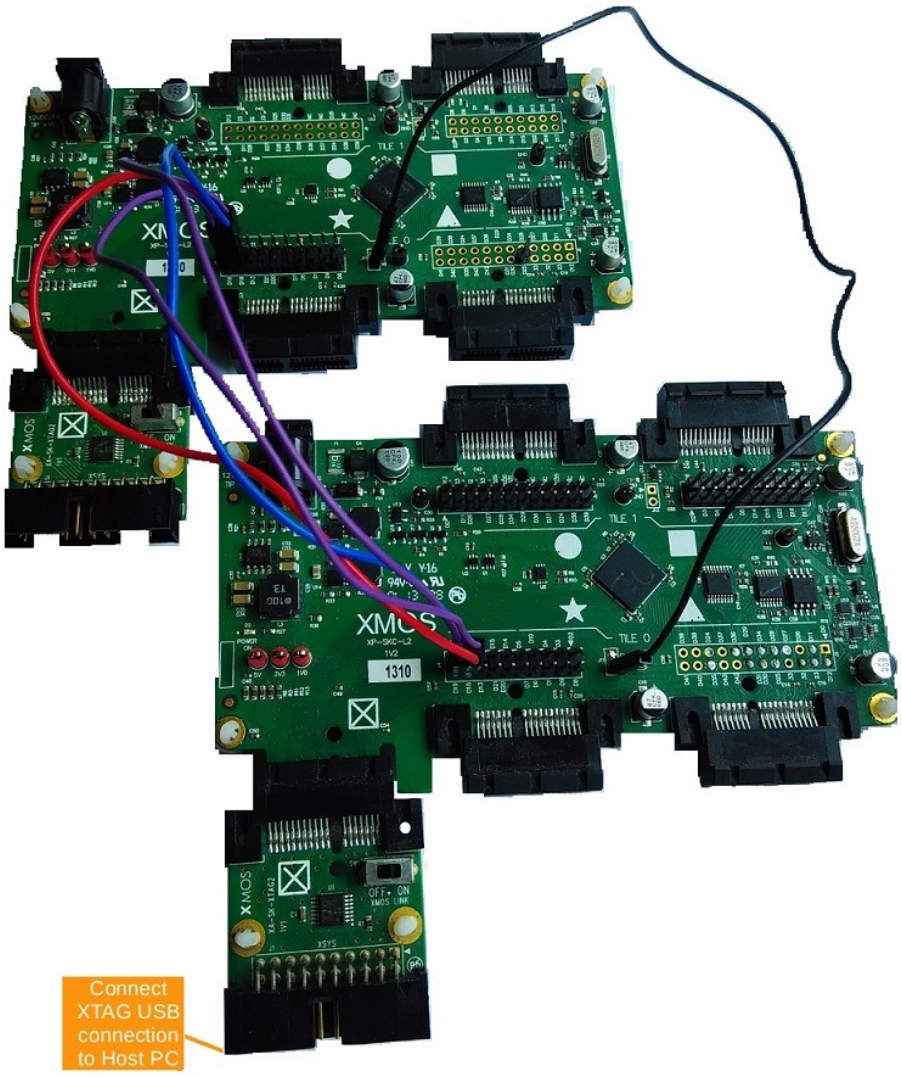

Figure 3: 2-wire XMOS xCORE-L16 sliceKIT set up for off-chip xCONNECT communication

To run the demo,

1. Connect xTAG-2 debug adapter to sliceKIT board 1
2. Connect the xTAG-2 to your development PC
3. Set the XMOS LINK to OFF on xTAG-2 debug adapter
4. Switch on the power supply to the sliceKIT core board
5. Repeat the above steps for sliceKIT board 2
6. Set the jumpers between the two sliceKITs as follows

Board	Tile	Slot	2w-pin1	2w-pin2	2w-pin3	2w-pin4	GND
sliceKIT 1	tile[0]	STAR	D16	D17	D18	D19	J4
sliceKIT 2	tile[0]	STAR	D19	D18	D17	D16	J4

APPENDIX B - Launching the demo device

Once the demo example has been built either from the command line using xmake or via the build mechanism of xTIMEcomposer studio you can execute the application on the sliceKIT core board.

Once built there will be a `bin` directory within the project which contains the binary for the xCORE device. The xCORE binary has a XMOS standard .xe extension.

B.1 Launching from the command line

From the command line, use the `xrun` tool to download code to the xCORE device. Navigate to the bin directory of the project and execute the code on the xCORE microcontroller as follows:

```
> xrun --id 1 --io app_link_tests_b1.xe      <-- Download and execute the xCORE code on the board 1
> xrun --id 0 --io app_link_tests_b2.xe      <-- Download and execute the xCORE code on the board 2
```

Once this command is executed, you will see the following text in the `receive` application's console window:

```
Demo started...tile id: 8001

Got credit

Communication rate: 1652937 bytes per sec                       ==>     13 Mbit/sec
Communication rate: 4629628 bytes per sec                       ==>     37 Mbit/sec
Communication rate: 4629628 bytes per sec                       ==>     37 Mbit/sec
Communication rate: 4629628 bytes per sec                       ==>     37 Mbit/sec
Communication rate: 4629628 bytes per sec                       ==>     37 Mbit/sec
Communication rate: 4629629 bytes per sec                       ==>     37 Mbit/sec

Application control token count:                849
Receive timeouts:                               157

Timed out...

Got credit
```

B.2 Launching from xTIMEcomposer Studio

From xTIMEcomposer Studio, use the run mechanism to download code to xCORE device. Select the xCORE binary from the bin directory, right click and then follow the instructions below:

- Select **Run Configuration**.
- Click **Apply** and then **Run**.

When the processor has finished booting you will see the following text in the xTIMEcomposer console window corresponding to the `receive` application:

```
Demo started...tile id: 8001

Got credit

Communication rate: 1652937 bytes per sec                       ==>     13 Mbit/sec
Communication rate: 4629628 bytes per sec                       ==>     37 Mbit/sec
Communication rate: 4629628 bytes per sec                       ==>     37 Mbit/sec
Communication rate: 4629628 bytes per sec                       ==>     37 Mbit/sec
Communication rate: 4629628 bytes per sec                       ==>     37 Mbit/sec
Communication rate: 4629629 bytes per sec                       ==>     37 Mbit/sec

Application control token count:                849
Receive timeouts:                               157

Timed out...

Got credit
```

While the demo is running, remove the cables connected between the two boards. The communication rate drops to 0 bits/sec:

```
Communication rate: 0 bytes per sec                       ==>      0 Mbit/sec
Communication rate: 0 bytes per sec                       ==>      0 Mbit/sec
Communication rate: 0 bytes per sec                       ==>      0 Mbit/sec
```

Reconnect the cables between the links in order to resume the communication. Note that after a reconnection, there could still be 0 bits/sec transfer rate; it may be attributed to the transmit link waiting for its credit. Once it receives the transmit credit, peak communication rate is resumed.

APPENDIX C - FAQs

C.1 What is the desired functionality of an application layer ?

The application layer will need to be tolerant of packet loss, because re-transmission at the transport layer can?t be implemented at the required data rate. All packet routing will be done by the xLINK app server logical core. More details on the application server task implementation to route packets to the participating nodes will be available in a separate application note.

C.2 Why to use static links instead of dynamic links?

xCONNECT assumes that the signal is correct, any fake edge will cause two extra bits to appear and will cause trouble. For 5-bit mode, every extra edge will cause four extra bits; the END-OF-MESSAGE will be destroyed, or randomly introduced then the next header will be random. It will be randomly routed. The best way to use an xCONNECT is to: - not allow it to route and - run a software protocol on it that has both a CRC (for error detection) and a time-out (in case it flattened a few transitions)

Where as using a statically routed link, no header is required as the routing is *not* attempted, and all the traffic on this link will always go to a fixed channel end. Another protocol (application) task at both ends monitors the data, manages a small protocol on top, and then resets-disconnects-reconnects the link on an error.

C.3 What are the advantages of xCONNECT?

No additional hardware is needed for either the physical layer or the link layer. In the case of static routing, the concept of a global network is no more. In order to get the BER down, use xCONNECT over a differential pair.

C.4 A few design parameters to be considered while evaluating xCONNECT

A 2-wire xCONNECT with 7.5ns symbol delay can achieve 106 MBit/s. A 7.5ns symbol delay corresponds to a toggle rate of 133 MHz. In order to design a more robust electrical link, and based on the required length of the traces, an LVDS transceiver may be needed.

In general, the following should assessed:

- throughput required
- what is the latency needed
- What is the acceptable number of lines
- What is the preferred Physical layer (LVDS, CMOS, and Ethernet)
- suitability of a CAN if the bw is ok
- suitability of a custom RS485

C.5 Modifying the application timeout parameters

This section describes a few parameters in the code and impact of changing them in the code.

- Try changing the rx_loop timeout to a value more than tx task timeout and observe what happens? when the tx loses its credit, it waits for the rx loop time out so that it gets additional credit in order for the transmission to resume. When the reception is active, receive timeout should not happen as the timer is loaded by the receive loop; in this demo, a periodic timeout is used to avoid any additional instructions on the receiver loop to avoid any impact on the receive bandwidth. For optimal demo operation, keep rx timeout lesser than tx timeout.
- As per the XS1 Link performance specification, the acheivable bandwidth should be higher than

the one observed in this demo application. By removing exlicit control token tests as available in the test_ct() function in the receive application, and using a trap handler based exception handling improve performance.
- Refer to a separate application demo which implement an application layer server to route the data traffic to the nodes operating at a higher bandwidth.

C.6 Modifying the application to run on a different tile or target

In the receive application code, modify the below macro and re-build the application:

```
#define DEMO_TILE                       0
```

Identify the appropriate link pins for the modified slot and tile and connect them as detailed in the setup section. In the receive application, ensure to update the channel identifier for the modified tile on which the application is running.Similarly if the target is different from xCORE-L16 sliceKIT board, modify the link pins accordingly.

C.7 How to run the application using RJ45-type LVDS sliceCARD connectors

A RJ45-type sliceCARD is built in order to run the application using Ethernet connector cables. The sliceCARD uses RJ45 type connector and uses LVDS signalling. This sliceCARD uses 2-wire Link A instead of Link B as specified in the above demo. Hence modify both the transmit and receive applications to reflect this change and re-build the applications.

Existing applications use link B:

```
#define LINK_NUM    3
```

Modify it to use link A:

```
#define LINK_NUM    2
```

To set up 2-wire xCONNECT connection between using these sliceCARDs, connection is detailed below:

To setup the hardware,

1. Connect xTAG-2 debug adapter to sliceKIT board 1
2. Connect the xTAG-2 to your development PC
3. Set the XMOS LINK to OFF on xTAG-2 debug adapter
4. Switch on the power supply to the sliceKIT core board
5. Repeat the above steps for sliceKIT board 2
6. Connect a cross-over cable between the two sliceCARDs

In order to run the demo, follow the instructions as available in the Launching the demo device section above.

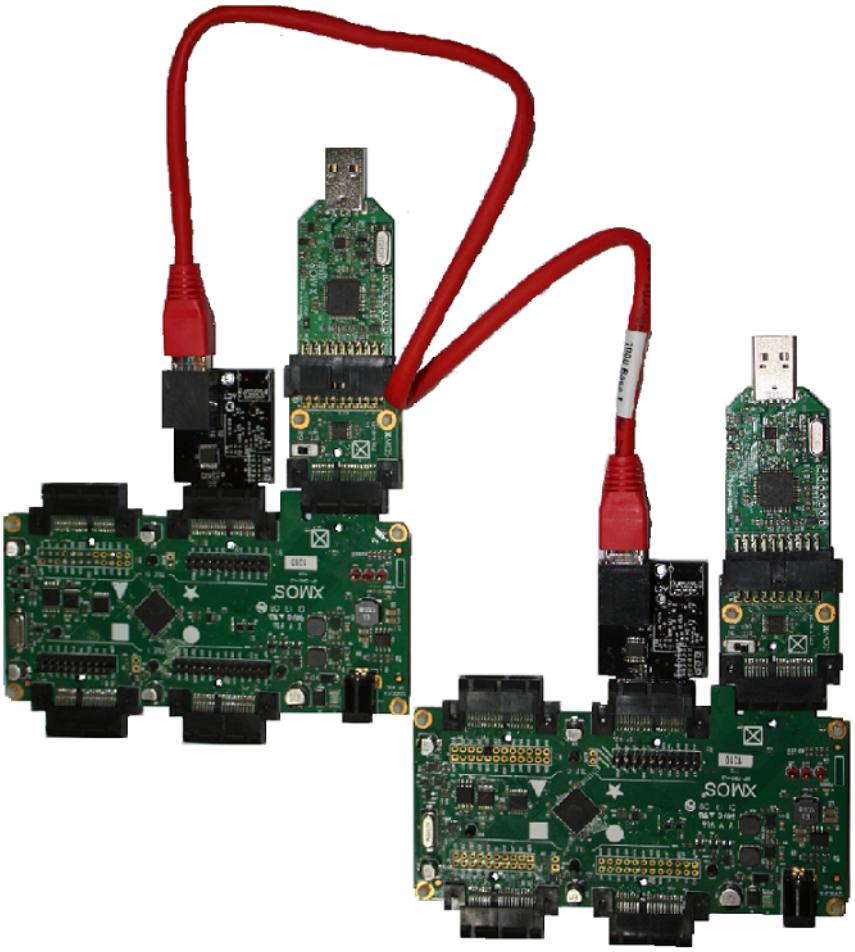

Figure 4: Two-wire xCONNECT communication set up using RJ45-type LVDS sliceCARD connected to xCORE-L16 sliceKITs

APPENDIX D - References

xTIMEcomposer User Guide

http://www.xmos.com/published/xtimecomposer-user-guide

xCONNECT Architecture

https://www.xmos.com/download/private/xCONNECT-Architecture%281.0%29.pdf

XS1 Link Performance and Design Guidelines

https://www.xmos.com/download/public/XS1-L-Link-Performance-Design-Guidelines%282.0%29.pdf

XS1-L System Specification

https://www.xmos.com/download/public/XS1-L-System-Specification(X1151D).pdf

sliceKIT hardware manual

https://www.xmos.com/download/private/sliceKIT-Hardware-Manual%281.0%29.pdf

XS1-L16A-128-FB324 Datasheet

https://www.xmos.com/download/private/XS1-L16A-128-FB324-Datasheet%281.1%29.pdf

XMOS xCORE Programming Guide

http://www.xmos.com/published/xmos-programming-guide

APPENDIX E - Full source code listing

E.1 Source code for transmit main.xc

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#include <platform.h>
#include <stdio.h>
#include <timer.h>
#include "link.h"
#include "chan.h"
#include "trycatch.h"

#define GET_SHARED_GLOBAL(x, g) asm volatile("ldw %0, dp[" #g "]":"=r"(x)::"memory")
#define SET_SHARED_GLOBAL(g, v) asm volatile("stw %0, dp[" #g "]"::"r"(v):"memory")

#define LINK_NUM    3
#define RE_ENABLE_TX_PERIOD 6
#define SEND_CTRL_TOKEN    2500000

int g_comm_state;

void re_enable_tx_link()
{
  while (1) {
    delay_seconds(RE_ENABLE_TX_PERIOD);
    SET_SHARED_GLOBAL(g_comm_state, 1);
    printf("Re-enable transmit link. Waiting for credit..\n");
    link_disable(LINK_NUM);
    link_enable(LINK_NUM);
  } //while (1)
}

void transmit_handler(int comm_state)
{
  unsafe streaming chanend c;
  int temp_state = 0;
  int err_ctr = 0;

  while (1) {
    GET_SHARED_GLOBAL(temp_state, g_comm_state);
    if (temp_state) {
      SET_SHARED_GLOBAL(g_comm_state, 0);
      comm_state = 4;
    }
    switch(comm_state) {
      case 0: //setup link direction
        int reg_val = 0;
        int direction = 0x1;
        reg_val = XS1_LINK_DIRECTION_SET(reg_val, direction);
        write_node_config_reg(tile[0], XS1_SSWITCH_SLINK_0_NUM + LINK_NUM, reg_val);
        comm_state = 1;
        break;
      case 1: //channel setup
        unsafe {
          c = chan_getr();
          chan_setd(c, 0x80030002);
        }
        comm_state = 2;
        break;
      case 2: /* reconfigure links, leaving only one open */
        for (int i=0; i<8; i++)
          link_disable(i);
        link_enable(LINK_NUM);
        comm_state = 3;
        break;
      case 3: /* Setup a static routing configuration */
        int ret = write_sswitch_reg(get_local_tile_id(), XS1_L_SSWITCH_XSTATIC_0_NUM + LINK_NUM, 0x80000000ul)
          ↪ ;
        delay_milliseconds(50);
        comm_state = 4;
        break;
      case 4: /* wait for transmit credits */
        do {
```

```
          link_reset(LINK_NUM);
          link_hello(LINK_NUM);
          delay_microseconds(100);
        } while (!link_got_credit(LINK_NUM));
        printf("Got credit\n");
        err_ctr = 0;
        comm_state = 5;
        break;
      case 5: /* send data tokens */
        /*if (!link_got_credit(LINK_NUM)) {
          comm_state = 4;
          break;
        }*/
        unsafe { c <: 'a'; }
        if (err_ctr++ == SEND_CTRL_TOKEN) {
          err_ctr = 0;
          unsafe {
            outct((chanend)c, XS1_CT_ACK);  //Send a control token as app sync point
          }
        }
        break;
      case 6: /* not used since this is a continuous running demo */
        chan_freer(c);
        link_disable(LINK_NUM);
        comm_state = 2;
        break;
      default:
        comm_state = 4;
        break;
    } //switch(comm_state)
  } //while (1)
}

void transmit_link()
{
  exception_t exception;

  TRY {
    printf("Demo started...tile id: %x\n",get_local_tile_id());
    transmit_handler(0);
  } CATCH (exception) {
    printf("Got exception.\n");
    transmit_handler(4);
  }
}

int main(void)
{
  par {
    on tile[0] : transmit_link();
    on tile[0] : re_enable_tx_link();
  }
  return 0;
}
```

E.2 Source code for receive main.xc

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#include <platform.h>
#include <stdio.h>
#include <assert.h>
#include <timer.h>
#include "link.h"
#include "chan.h"

#define LINK_NUM    3
#define RX_TIME_OUT_TICKS 500000000

#define DEMO_TILE           0

#if (DEMO_TILE == 0)
#define CHANEND_TILE_ID     0x80010002
#else
```

```
#define CHANEND_TILE_ID     0x80030002
#endif

#define GET_SHARED_GLOBAL(x, g) asm volatile("ldw %0, dp[" #g "]":"=r"(x)::"memory")
#define SET_SHARED_GLOBAL(g, v) asm volatile("stw %0, dp[" #g "]"::"r"(v):"memory")

int g_data_tokens;
int g_ctrl_tokens;
int g_timeout_cnts;

void reporting_task()
{
  timer bw_tmr;
  int bw_t;
  int i = 0;
  int j = 0;
  int k = 0;
  int full_rep_cnt = 0;

  bw_tmr :> bw_t;

  while (1) {
    select {
      case bw_tmr when timerafter(bw_t + 100000000) :> void:
        GET_SHARED_GLOBAL(i, g_data_tokens);
        GET_SHARED_GLOBAL(j, g_ctrl_tokens);
        GET_SHARED_GLOBAL(k, g_timeout_cnts);
        printf("Communication rate: %d bytes per sec \t\t\t ==>\t %d Mbit/sec\n", i, ((i*8)/1000000));
        if (!(full_rep_cnt++ % 10)) {
          printf("\nApplication control token count: \t\t%d  \n", j);
          printf("Receive timeouts: \t\t\t\t%d \n\n", k);
        }
        SET_SHARED_GLOBAL(g_data_tokens, 0);
        bw_tmr :> bw_t;
        break;
    }
  }
}

#pragma select handler
void test_ct(unsafe streaming chanend c, int &j)
{
  char r = 'z';
  unsafe {
    if (testct((chanend)c)) {
      r = inct((chanend) c);
      j++;
    }
    else {
      c :> r;
      unsafe {
        volatile int * unsafe p = &g_data_tokens;
        *p = *p + 1;
      }
    }
  }
} //end of test_ct

void receive_link()
{
  unsafe streaming chanend c;
  timer rx_loop_tmr;
  int t;
  int ctrl_tkn_ctr = 0;
  int tm_out_ctr = 0;
  int rx_loop = 0;
  int comm_state = 0;

  rx_loop_tmr :> t;

  printf("Demo started...tile id: %x\n",get_local_tile_id());
  while (1) {
    switch (comm_state) {
      case 0: //setup link direction
        int reg_val = 0;
```

```
        int direction = 0x0;
        reg_val = XS1_LINK_DIRECTION_SET(reg_val, direction);
        write_node_config_reg(tile[1], XS1_SSWITCH_SLINK_0_NUM + LINK_NUM, reg_val);
        comm_state = 1;
        break;
      case 1: //channel alloc
        unsafe {
          c = chan_getr();
          //printf("Allocated chanend: %x\n", c);  //debug print, if the tile is changed
          assert((int)c == CHANEND_TILE_ID);
        }
        comm_state = 2;
        break;
      case 2:  //reconf links, leaving only one open
        for (int i=0; i<8; i++)
            link_disable(i);
        link_enable(LINK_NUM);
        /* setup static link for the communicating link */
        int ret = write_sswitch_reg(get_local_tile_id(), XS1_L_SSWITCH_XSTATIC_0_NUM + LINK_NUM, 0x80030000ul)
          ↪ ;//TODO:
        delay_milliseconds(50);
        comm_state = 3;
        break;
      case 3: //wait for transmit credits
        do {
          link_reset(LINK_NUM);
          link_hello(LINK_NUM);
          delay_microseconds(100);
        } while (!link_got_credit(LINK_NUM));
        printf("\nGot credit\n\n");
        /* setup local control variables */
        rx_loop = 1;
        tm_out_ctr++;
        rx_loop_tmr :> t;
        comm_state = 4;
        break;
      case 4: //receive data loop
        char r = 'z';
        while (rx_loop) {
          unsafe {
            select {
              case test_ct(c, ctrl_tkn_ctr):
                rx_loop_tmr :> t;
                break;
              //case c :> j: break; // breaks at unexpected control tokens
              case rx_loop_tmr when timerafter(t + RX_TIME_OUT_TICKS) :> void:
                SET_SHARED_GLOBAL(g_ctrl_tokens, ctrl_tkn_ctr);
                SET_SHARED_GLOBAL(g_timeout_cnts, tm_out_ctr);
                printf("\nTimed out...\n\n");
                rx_loop = 0;
                comm_state = 3;
                break;
            }
          }
        } //while (rx_loop)
        break;
      case 6: //end of communication; not used in this demo since its a continuous run
        chan_freer(c);
        link_disable(LINK_NUM);
        comm_state = 1;
        break;
      default:
        break;
    } //switch (comm_state)
  } // while (1)
}

int main(void)
{
  par {
    on tile[DEMO_TILE] : receive_link();
    on tile[DEMO_TILE] : reporting_task();
  }
  return 0;
}
```