Application Note: AN10129

Using zip/unzip to implement streaming interfaces on multibit ports

This application note shows how to implement a streaming interface like I2S using a buffered 4-bit port. Streaming interfaces are usually implemented using a buffered 1-bit port for best performance. The ability to use 4-bit ports to stream multiple independent 1-bit signals provides added flexibility when all the 1-bit ports are utilized.

xCORE-200 devices feature zip and unzip instructions, which facilitate the efficient use of multibit ports as multiple single bit ports.

This examples shows how to send and receive four 1-bit streams of data. However, instead of using four 1-bit ports, a single 4-bit port will be used, treating each pin of the port as if it were a 1-bit port.

The new zip and unzip instructions of xCORE-200 provide enough performance to enable four I2S interfaces at 192 kHz sampling rate over 4-bit ports.

Required tools and libraries

- xTIMEcomposer Tools - Version 14.1.0

Required hardware

This application note is designed to run on any XMOS multicore microcontroller or the XMOS simulator.

Prerequisites

None

Copyright 2018 XMOS Ltd.

1 Overview

1.1 Introduction

To transmit data serially on a specific pin of a multibit port, the serial data has to be transformed. Any nibble that is output on a 4-bit port has to be composed of bits from each of the four serial streams.

- Serial stream 0 is transmitted on bit 0 of the 4-bit-port.
- Serial stream 1 is transmitted on bit 1 of the 4-bit-port.
- Serial stream 2 is transmitted on bit 2 of the 4-bit-port.
- Serial stream 3 is transmitted on bit 3 of the 4-bit-port.

See Chapter 6.1 Ports in[1] for more information on I/O Ports

The zip instruction can be used to efficiently generate the output data for the multibit port. The unzip instruction is used for the reverse transformation.

The example uses the port loopback feature of the xSIM simulator to connect the input ports to the output ports and implement a HW loopback test.

1.2 zip/unzip of data

The zip and unzip instructions are documented in XMOS XS2 Architecture (ISA)[2].

Like closing a zipper, chunks from two words are alternately joined into a 64-bit result by the zip instruction. The size of the chunks (or zipper teeth) can be configured. Like opening a zipper, unzip is the reverse operation taking 64-bit data as an input and producing two unzipped 32-bit results.

The zip operation to transform the data from four streaming channels for output on a 4-bit port is illustrated in Figure 1:

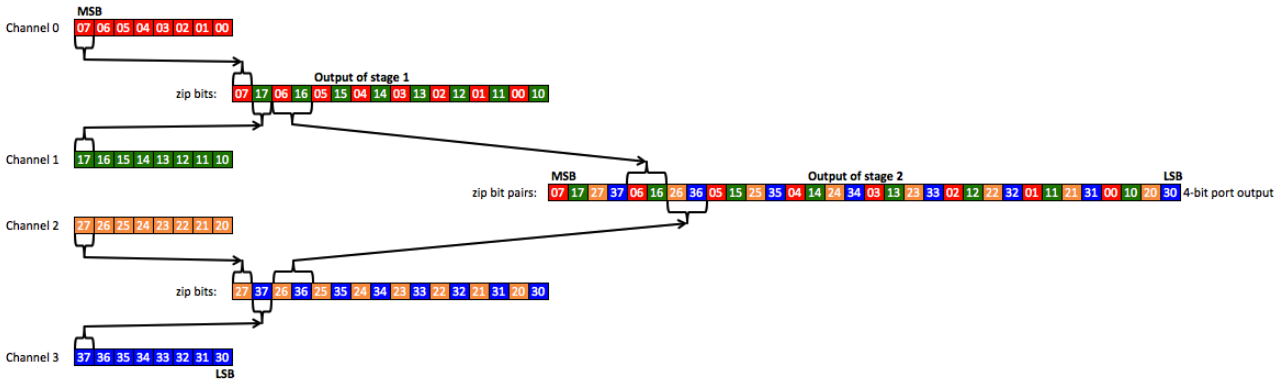

Figure 1: Zip operations to transform four serial streams for output on a 4-bit port

Zipping is done in two stages. Note that the first stage uses a chunk size of one bit. The second stage uses a chunk size of two bits.

[1] http://www.xmos.com/published/xmos-programming-guide
[2] http://www.xmos.com/published/xs2-isa-specification

Note: To keep the image in Figure 1 readable, the illustration is limited to zipping four bytes. In reality, zip instructions have word operands.

The function zip_4_streams_for_4_bit_port transforms four words. This requires a total of four zip instructions. Two for each zip stage:

```
// zip 4 words (streams) of serial data for output on a 4-bit port
// word 0 will be output on bit0 of the 4-bit port
// word 1 will be output on bit1 of the 4-bit port
// word 2 will be output on bit2 of the 4-bit port
// word 3 will be output on bit3 of the 4-bit port
static inline void zip_4_streams_for_4_bit_port(unsigned int outputs[4]) {
    unsigned long long tmp64_0, tmp64_1;

    // zip x, y, s.
    // zip packs of 1 bit. MSB of x comes first.
    tmp64_0 = zip(outputs[0], outputs[1], 0);
    tmp64_1 = zip(outputs[2], outputs[3], 0);

    outputs[0] = (unsigned int) tmp64_0;
    outputs[1] = (unsigned int) tmp64_1;
    outputs[2] = (unsigned int) (tmp64_0 >> 32);
    outputs[3] = (unsigned int) (tmp64_1 >> 32);

    // zip packs of two bits!
    tmp64_0 = zip(outputs[0], outputs[1], 1);
    tmp64_1 = zip(outputs[2], outputs[3], 1);

    outputs[0] = (unsigned int) tmp64_0;
    outputs[1] = (unsigned int) (tmp64_0 >> 32);
    outputs[2] = (unsigned int) tmp64_1;
    outputs[3] = (unsigned int) (tmp64_1 >> 32);

    //MSB from word 0 (original output[0]) is now MSB of outputs[3]!

}
```

Note: The function unzip_from_4_bit_port_to_4_streams does the reverse operation.

Copyright 2018 XMOS Ltd.

2 Integration with an existing streaming interface

The following describes the integration of the zip/unzip functionality with an existing streaming interface like I2S.

2.1 Output:

Call the function zip_4_streams_for_4_bit_port to transform 4 words of serial data into 4 words formatted to be output successively on a 4-bit port:

```
// zip output data
zip_4_streams_for_4_bit_port(outputs);
```

Note that the most significant data will end up in outputs[3], the least significant data in outputs[0]. So to output the MSBs of the four streams first, outputs[3] has to be output first (bit reversed) to the 4-bit port:

```
// Formats like I2S transmit MSB first. bitrev is needed because port outputs LSB first
p_4bit_out <: bitrev(outputs[3]);
```

2.2 Input:

Call the function unzip_from_4_bit_port_to_4_streams to transform 4 words received from a 4-bit port into 4 words of serial data. We assume again a protocol like I2S where the MSB is transmitted first. That means the received data has to be bit reversed:

```
p_4bit_in :> tmp;
inputs[3] = bitrev(tmp);
```

When 4 words are received, call the unzip function to transform the data back into 4 words of serial data:

```
// unzip input data
unzip_from_4_bit_port_to_4_streams(inputs);
```

Copyright 2018 XMOS Ltd.

3 Performance

The example was tested on the simulator with a port clock (bit clock) of 12.5 MHz. The bitclock of an I2S interface running at 192kHz sampling rate would be 12.288 MHz so this would work.

Copyright 2018 XMOS Ltd.

APPENDIX A - Running the example

To make the data loopback work, the:

```
out buffered port:32 p_4bit_out = XS1_PORT_4A;
```

is connected to the:

```
in buffered port:32 p_4bit_in = XS1_PORT_4C;
```

using the port loopback feature of the xSIM simulator.

After the data is output on XS1_PORT_4A and looped back to XS1_PORT_4C it is unzipped and then checked for correctness. The received data and the test result are printed on the console. The messages should look like this:

```
unzipped input data:
87654321
18765432
21876543
32187654
Loopback data check PASS :)
```

A.1 Launching from the command line

From the command line the xSIM simulator is launched. The port loopback plugin is activated connecting the XS1_PORT_4A (output) to the XS1_PORT_4C (input).

```
xsim --plugin LoopbackPort.dll '-port tile[0] XS1_PORT_4A 4 0 -port tile[0] XS1_PORT_4C 4 0'
  ↪ AN10129_xCORE_200_using_zip_unzip_with_multibit_ports.xe
```

See[3] for more information about using port loopback with the xSIM simulator on the command line.

[3] http://www.xmos.com/support/examples/AN10098

A.2 Launching from xTIMEcomposer Studio

From xTIMEcomposer Studio, create a run or debug application. Click on the .xe file in the bin directory and select Run->Debug Configurations. Double click on "xCORE Application" to create a new Debug Configuration.

Click Device options "simulator" and then click on the Simulator tab. To connect the signal loopback, click "Enable pin connections" and then add. Select from: tile[0] XS1_PORT_4A and then to: tile[0] XS1_PORT_4C as shown here:

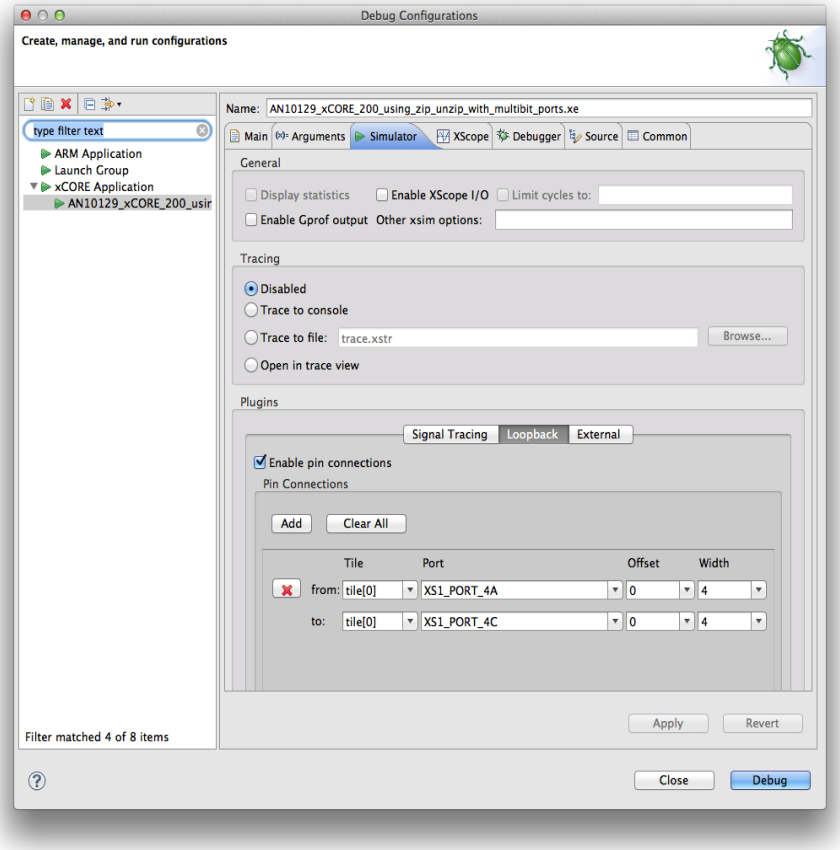

Figure 2: Debug configurations dialog. Simulator tab showing Signal Loopback connections

Click "Apply" and then "Debug".

Now you should see the expected console output shown in §A.

A.3 Waveform Viewer

One advantage of §A.2 is the Waveform Viewer. It is useful to analyse the signal activity on the 4-bit ports. For further information on using the Waveform Viewer please consult AN10102 How to enable VCD tracing when running on the simulator[4].

[4] https://www.xmos.com/support/examples/AN10102

APPENDIX B - 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

XMOS XS2 Architecture (ISA)

http://www.xmos.com/published/xs2-isa-specification

APPENDIX C - Full source code listing

C.1 Source code for main.xc

```
// Copyright (c) 2015-2016, XMOS Ltd, All rights reserved

/**
xCORE-200 devices contain support for zip and unzip instructions, which
facilitate the efficient use of multibit ports as multiple single
bit ports.

This examples shows how to send and receive four 1-bit streams of data.
However, instead of using four 1-bit ports, a single 4-bit port will be used,
treating each pin of the port as if it were a 1-bit port.

Two 4-bit ports are defined, an output and an input port. The output port
will be connected to the input port using the loopbacks in the simulation
environment. We will use the zip instruction to format the data correctly,
then send it to the output port. We will then read it back from the input port and
use unzip to recreate the original data. It will then be automatically checked and
displayed on the console.

**/


#include <xs1.h>
#include <stdio.h>
#include <xclib.h>

out buffered port:32 p_4bit_out = XS1_PORT_4A;
in buffered port:32 p_4bit_in = XS1_PORT_4C;
out port p_bbit_clock = XS1_PORT_1A;

clock bit_clock = XS1_CLKBLK_1;

//#define SW_LOOPBACK
#define DEBUG_PRINT

// zip 4 words (streams) of serial data for output on a 4-bit port
// word 0 will be output on bit0 of the 4-bit port
// word 1 will be output on bit1 of the 4-bit port
// word 2 will be output on bit2 of the 4-bit port
// word 3 will be output on bit3 of the 4-bit port
static inline void zip_4_streams_for_4_bit_port(unsigned int outputs[4]) {
    unsigned long long tmp64_0, tmp64_1;

    // zip x, y, s.
    // zip packs of 1 bit. MSB of x comes first.
    tmp64_0 = zip(outputs[0], outputs[1], 0);
    tmp64_1 = zip(outputs[2], outputs[3], 0);

    outputs[0] = (unsigned int) tmp64_0;
    outputs[1] = (unsigned int) tmp64_1;
    outputs[2] = (unsigned int) (tmp64_0 >> 32);
    outputs[3] = (unsigned int) (tmp64_1 >> 32);

    // zip packs of two bits!
    tmp64_0 = zip(outputs[0], outputs[1], 1);
    tmp64_1 = zip(outputs[2], outputs[3], 1);

    outputs[0] = (unsigned int) tmp64_0;
    outputs[1] = (unsigned int) (tmp64_0 >> 32);
    outputs[2] = (unsigned int) tmp64_1;
    outputs[3] = (unsigned int) (tmp64_1 >> 32);

    //MSB from word 0 (original output[0]) is now MSB of outputs[3]!

}
```

```
// unzip from 4 words received from a 4-bit into 4 words (streams)
static inline void unzip_from_4_bit_port_to_4_streams(unsigned int inputs[4]) {
    unsigned long long tmp64_0, tmp64_1;
    // make long longs
    tmp64_0 = (unsigned long long) (inputs[1]) << 32 | inputs[0];
    tmp64_1 = (unsigned long long) (inputs[3]) << 32 | inputs[2];

    // unzip into packs of 2 bits
    {inputs[0], inputs[1]} = unzip(tmp64_0, 1);
    {inputs[2], inputs[3]} = unzip(tmp64_1, 1);

    // make long longs
    tmp64_0 = (unsigned long long) (inputs[2]) << 32 | inputs[0];
    tmp64_1 = (unsigned long long) (inputs[3]) << 32 | inputs[1];

    // unzip into packs of 1 bits
    {inputs[0], inputs[1]} = unzip(tmp64_0, 0);
    {inputs[2], inputs[3]} = unzip(tmp64_1, 0);
}

#pragma unsafe arrays // Disable array range check for better performance
int main() {

  unsigned int outputs[4] = {0x87654321, 0x18765432, 0x21876543, 0x32187654};
  unsigned int inputs[4];

  unsigned expected_data[4];

  unsigned int tmp;

  unsigned int t;

  // generate expected data
  for(int i=0; i<4; i++) {
      expected_data[i] = outputs[i];
  }

  // setup clock
  configure_clock_rate(bit_clock, 100, 8); // 12.5 MHz
  // Note: This proves that a 12.288 MHz bbit_clock works. That's 4 I2S stereo channels at 192kHz
  // 25MHz is too fast : configure_clock_rate(bit_clock, 100, 4); // 25 MHz

  configure_out_port(p_4bit_out, bit_clock, 0);
  configure_in_port(p_4bit_in, bit_clock);
  //configure_clock_out

  start_clock(bit_clock);

#ifdef DEBUG_PRINT
  printf("\noutput data:\n");
  for(int i=0; i<4; i++) {
      printf("%0.8x\n", outputs[i]);
  }
#endif

  // zip output data
  zip_4_streams_for_4_bit_port(outputs);

#ifdef DEBUG_PRINT
  printf("\nzipped output data:\n");

  for(int i=0; i<4; i++) {
      printf("%0.8x\n", outputs[i]);
  }
#endif
```

```
  // align port timing
  p_4bit_out <: 0 @ t;
  t += 100;
  // output 0 at t
  p_4bit_out @ t <: 0;
  // To align input with output, set the time of the next input on p_4bit_int.
  asm volatile("setpt res[%0], %1"::"r"(p_4bit_in),"r"(t+15));


  // MSB of channel 0 is MSB of outputs[3]!
  // See zip_4_streams_for_4_bit_port for details

  // Formats like I2S transmit MSB first. bitrev is needed because port outputs LSB first
  p_4bit_out <: bitrev(outputs[3]);
  p_4bit_in :> tmp;
  inputs[3] = bitrev(tmp);

  p_4bit_out <: bitrev(outputs[2]);
  p_4bit_in :> tmp;
  inputs[2] = bitrev(tmp);

  p_4bit_out <: bitrev(outputs[1]);
  p_4bit_in :> tmp;
  inputs[1] = bitrev(tmp);

  p_4bit_out <: bitrev(outputs[0]);
  p_4bit_in :> tmp;
  inputs[0] = bitrev(tmp);

#ifdef SW_LOOPBACK
  for(int i=0; i<4; i++) {
      inputs[i] = outputs[i];
  }
#endif

  // unzip input data
  unzip_from_4_bit_port_to_4_streams(inputs);

#ifdef DEBUG_PRINT
  printf("\nunzipped input data:\n");
  for(int i=0; i<4; i++) {
      printf("%0.8x\n", inputs[i]);
  }
#endif

  // check the data
  unsigned errors=0;
  for(int i=0; i<4; i++) {
      if(inputs[i] != expected_data[i]) {
          printf("ERROR in input data %d. Expected %x, Received %x\n", i, expected_data[i], inputs[i]);
          errors++;
      }
  }
  if(errors==0) {
      printf("Loopback data check PASS :)\n");
  } else {
      printf("Loopback data check FAIL, %d Errors\n",errors);
  }

  return 0;
}
```

XMOS®
Copyright © 2018, All Rights Reserved.

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.