Application Note: AN01021

Interfacing High Speed ADCs with xCORE

The application notes shows how to interface high speed ADCs with xCORE devices. It also gives an overview about the advantages of using xCORE buffered I/O ports, clock blocks and xCONNECT communication channels.

The code associated with this application note povide an example to interface Texas Instruments high speed ADC (ADS7863A) to xCORE devices. The application note contains simulation that shows how the data is sampled by the xCORE device.

Required tools and libraries

- xTIMEcomposer Tools - Version 13.2

Required hardware

This application note is designed to run using xSIM (XMOS simulator tool). No additional hardware is required to run the example.

The example code provided with the application has been implemented and tested using the XMOS simulator tool (xSIM) and can run on any xCORE device.

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].

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

1 Overview

This application notes shows how to interface an xCORE device with an external ADC like TI ADS7863A. To interface the ADC with an xCORE device, one logical core is sufficient. But, this application notes uses three logical cores for demonstrating xCORE features. One logical core (master_ads7863a) is used to interface with the ADC using xCORE I/O ports, one logical core is for the user application (app) – receives data that is sent by the master_ads7863a logical core – and one logical core to emulate the ADS7863A device (slave_side_ads7863a_simulation). This application note refers to the ADV in Mode-I; information about other modes supported by this ADC is available in the *ADS7863A data sheet*.[2]

The timing diagram of the ADC is shown below:

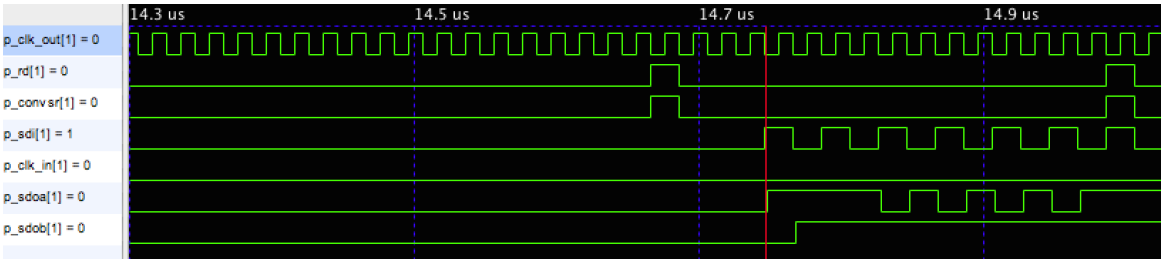

Figure 1: Timing diagram of ADC

1.1 Applications where high speed ADCs are used

- Motor Control
- Multi-Axis Positioning Systems
- Three-Phase Power Control
- Sensor data acquisition

[2] http://www.ti.com/lit/ds/symlink/ads7863a.pdf

1.2 Block Diagram

The block diagram (Figure 2) shows the ADC interface with an xCORE device in Mode-I.

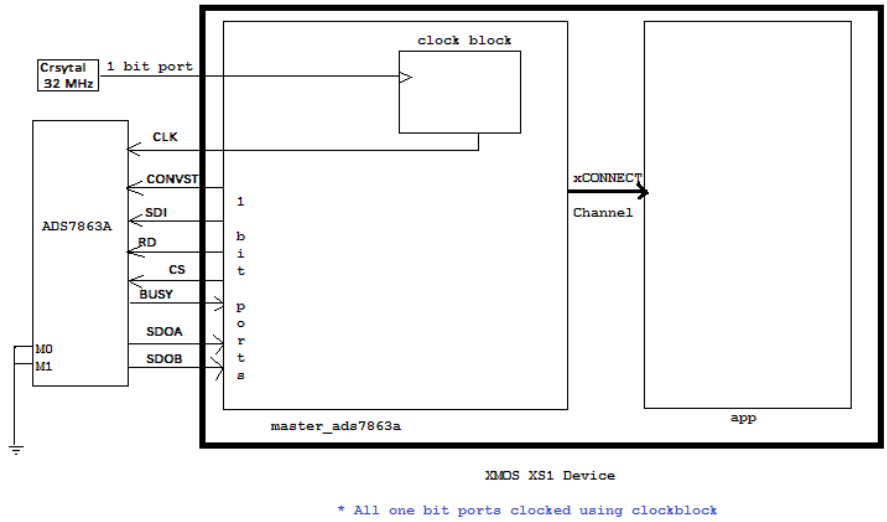

Figure 2: Block digaram of ADC interface with xCORE

2 Application overview

The example in this document does not have any external dependancies on application libraries other than those supplied with the XMOS development tools.

This example is implemented using three logical cores which perform the following functionality

- One logical core (master_ads7863a) is used to interface with the ADC using xCORE I/O ports and generate clock for the ADC and read the ADC data when there is new data available.
- One logical core (slave_side_ads7863a_simulation) is used in this example to emulate the ADC slave.
- One logical core (app) is used as an application core to receive the ADC data using a channel.

This can be seen in the following core diagram

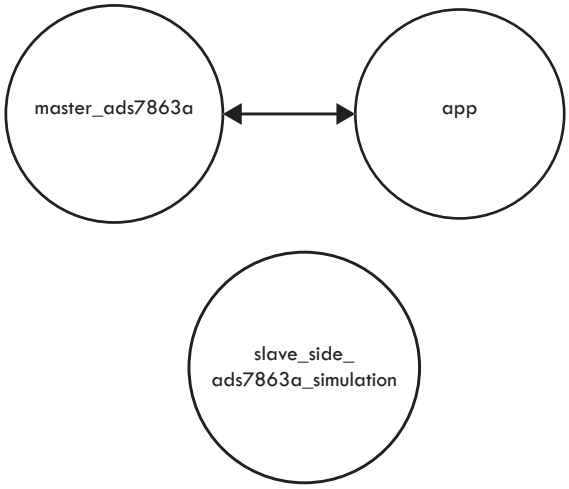

Figure 3: Core diagram showing communication between logical cores

2.1 Advantages of using xCORE technology

Unlike traditional microcontrollers, xCORE devices have multiple logical cores which run several tasks independently in parallel and have high I/O flexibility. xCORE I/Os have less latency and can respond to events very quickly compared to traditional microcontrollers, so they are well suited for time critical and high speed data acquisition applications. More information is available in the *xCORE Architecture Introduction*[3] document.

2.1.1 xCORE I/O ports

xCORE I/O ports have unique functionality that allows a reference clock to be used to synchronize the input and output operations on a port. The ports can be configured to buffer the transfers, and can be easily converted between serial and parallel form. Ports can be configured to wait for a condition and then generate events that trigger the logical processing core. All of these features are used in this application. More information on ports is available in the *XS1 Ports Introduction*[4].

[3] http://www.xmos.com/published/xcore-architecture
[4] http://www.xmos.com/published/xs1-ports-introduction

2.1.2 xCONNECT channels

xCORE devices can communicate between the logical cores using high speed, low latency channels. There are two types of channels: streaming channel and normal channel. This application uses streaming channels which have less overhead compared to normal channels. They are suitable for high speed communication between logical cores, but the number of streaming channels you can have between the tiles is limited. See the *xCONNECT Architecture*[5] document for more details.

2.2 Application Description

The application uses nine 1bit ports and a clock block to interface with ADC. It also uses four 1bit ports and one clock block to emulate the ADC slave. The ADC requires a 32MHz clock to be generated by the xCORE device. The I/O ports on xCORE devices generally run synchronous to the 100MHz reference clock. The ports can be configured to run at different clock speeds by dividing the reference clock by multiples of 2, but this still does not deliver the required 32MHz clock. Therefore, the application must use an external 32MHz oscillator that's connected to a 1bit port, as shown in *Figure 2*. For the demonstration, a 50MHz clock was used. The clock block is clocked from the crystal clock signal using the input port. All the other input and output ports are set to clock based on the clock block, so the data will be sampled synchronous to the clock block at the speed of 50MHz. More information on programming clock blocks and buffered ports see the *XMOS Programming Guide*[6].

2.3 PortMap details

In the xCORE architecture, a Port connects a processor to one or more physical pins. The portMap defines which ports are assigned to which software functions. So in this example, clock signal, conversion start, data ready, data input, read channel A data and read channel B data are required to interface with TI ADS7863A ADC device. ADC conversion starts when the conversion start pin goes high. Converted ADC data is read using sdoa pin for channel A and sdob pin for channel B. sdi pin is for sending configuration information to the ADC (specifies the settings for conversion). The data read from the ADC after conversion using sdoa and sdob is 12 bit. Data is available for reading only after two clock cycles after ready signal goes low.

xCORE ports required to interface with TI ADS7863A ADC are listed below:

Port Name	Use
XS1_CLK_BLK_1	Clockblock to generate 50MHz clock as 32 MHz cannot be generated by xCORE internally.
p_clk_in	Input port to input 32MHz clock signal to clock block clk; Not used in this simulation as the demonstration uses a 50MHz internal clock.
p_clk_out	Output port to driver clock to the ADS7863A device.
p_rd	Output port for read write signal to the ADS7863A device.
p_convsr	Output port to send the start conversion signal to the ADC.
p_sdi	Output port to send configuration information to the ADC.
p_cs	Output port for chip select (enable and disable the ADC).
p_sdoa	Input port for reading Channel A conversion results from the ADC.
p_sdob	Input port for reading Channel B conversion results from the ADC.
p_busy	Input port for reading the status of the ADS7863A device.

[5] http://www.xmos.com/published/xconnect-architecture
[6] http://www.xmos.com/published/xmos-programming-guide

Details about the functionality of the ports are available in the TI *ADS7863A datasheet*[7].

2.4 PortMap for Slave Emulation

portmap for emulating the slave device is shown below

Port Name	Use
p_dummy_1	Output port to send Channel A converted values.
p_dummy_2	Output port to send Channel B converted values.
p_signal	Output port to indicate the status of the ADC.
p_clk_in_sim	Input port to receive clock.
clk_1	Clock block that is configured to run based on the input clock signal.

2.5 The application main() function

The main() function is contained in main.xc as follows:

```
int main()
{
    streaming chan c;
    par
    {
        on tile[0] : master_ads7863a(c);
        on tile[0] : slave_side_ads7863a_simulation();
        on tile[0] : app(c);
    }
}


//End
```

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

- Three logical cores are running in parallel on tile 0.
- The application and master logical cores communicate the ADC information using a channel c.

[7] http://www.ti.com/lit/ds/symlink/ads7863a.pdf

2.6 Master_ads7863a() logical core

The logical core master_ads7863a is available in the main.xc file as follows:

```
void master_ads7863a( streaming chanend c)
{
  unsigned data_1,data_2,ts;
  //reference clock is set to 50MHz
  configure_clock_rate(clk,100,2);
  configure_port_clock_output(p_clk_out, clk);
  //Configuring all the ports synchronous to reference clock
  configure_out_port(p_convsr, clk, 0);
  configure_out_port(p_sdi, clk, 0);
  configure_out_port(p_rd, clk, 0);
  configure_in_port(p_sdoa, clk);
  configure_in_port(p_sdob, clk);

  p_rd<:0x0000000;
  p_convsr<:0x00000000;
  p_sdi<:0x00000000;
  set_port_shift_count(p_sdoa,16);
  set_port_shift_count(p_sdoa,16);
  start_clock(clk);
  p_cs<:0;
  while(1)
  {
      p_rd @ (ts+5)<:0x00010001 ;
      p_convsr @ (ts+5) <:0x00010001;
      //write LSB first for two channels at a time
      p_sdi @ (ts+7) <:0x05555550;
      //Read two channels at a time. As we are usign 32 bit buffered ports
      p_sdoa:> data_1;
      p_sdob:> data_2 @ ts;
      //Send the data read to the application using channels
      c<:data_1;
      c<:data_2;
  }
}
```

Looking at the logical core in more detail, you can see the following:

- Clock block clk is set to run at 50MHz.
- Generated 50MHz clock is outputted on port p_clk_out. The clock output is used to input clock to the ADC.
- Output ports p_convsr, p_sdi, p_rd are configured to run synchronously using clock block clk. Information on timed output on a port is available in the *how to perform timed output on a port*[8] document.
- ADC requires conversion start signal along with the clock input. After receiving the conversion start signal, the ADC starts the conversion.
- Configuration of ADC is done using p_sdi. The conversion start should be enabled once the configuration is set.
- The input ports p_sdoa and p_sdob are clocked using clock block clk and read the converted ADC data after each conversion.

[8] https://www.xmos.com/download/public/app_port_timed_output_example-README(1.1.1rc0.a).pdf

- p_rd and p_convsr ports signals are made high for every 16 clock pulses to indicate start conversion to the ADC.
- Data from Channel A is read and sent to the application using a channel c
- Data from Channel B is read and sent to the application using a channel c.
- As the application uses a 32 bit buffered port, two samples must be read together. Information on buffered output ports is available in the *How to use buffering for port output*[9] document.

2.7 slave_side_ads7863a_simulation() logical core

The logical core slave_side__ads7863a_simulation is available in the main.xc file as follows:

```
void slave_side_ads7863a_simulation()
{
    unsigned data;
    configure_clock_rate(clk_1,100,2);
    configure_out_port(p_dummy_1, clk_1,0);
    configure_out_port(p_dummy_2, clk_1,0);
    configure_in_port(p_signal, clk_1);
    start_clock(clk_1);

    while(1)
    {
        // wait for conversion start signal from Master
        p_signal:>data ;

        if(data)
        {
            clearbuf(p_dummy_1);
            clearbuf(p_dummy_2);
            //Output Channel A and CHannel B information. Two samples at a time.
            p_dummy_1<: 0x3AAA3AAF ;
            p_dummy_2<: 0xFFFFFFFE;

        }
    }
}
```

Looking at the logical core in more detail, you can see the following:

- clockblock clk_1 is configured to run according to the clock signal.
- Output ports p_dummy_1, p_dummy_2 are configured to run synchronous to clock block clk_1.
- The logical core reads data on p_signal port and sends data on the ports p_dummy_1, p_dummy_2 emulating the slave ADC device.
- The conversion start signal of master_ads7863a logical core is connected to the p_signal port.
- When the master_asd7863a core sends a conversion start signal, the p_signal port detects and sends data back to the master_ads7863a core using ports p_dummy_1 and p_dummy_2.

[9] https://www.xmos.com/download/public/app_port_buffered_output_example-README(1.1.1rc0.a).pdf

2.8 app() logical core

The logical core app is available in the main.xc file and is as follows,

```
void app( streaming chanend c)
{
    unsigned data;
    while(1)
    {
        //Read Channel A and CHannel B data alternatively.
        c:>data;
    }
}
```

Looking at the logical core in more detail, you can see that: channel c is waiting for the ADC channel number; data is read from the channel A and channel B alternatively.

3 Launching the demo application

Once the demo application is built using the xTIMEcomposer Studio, the example can be run on the simulator using the following steps: -0.5em

1. Right click on the binary in the `bin` directory within the project and select **Run as ▶ Run configurations**. This will open up *Run configurations* menu.

2. Select **Run on Simulator** option as shown in Figure 4:

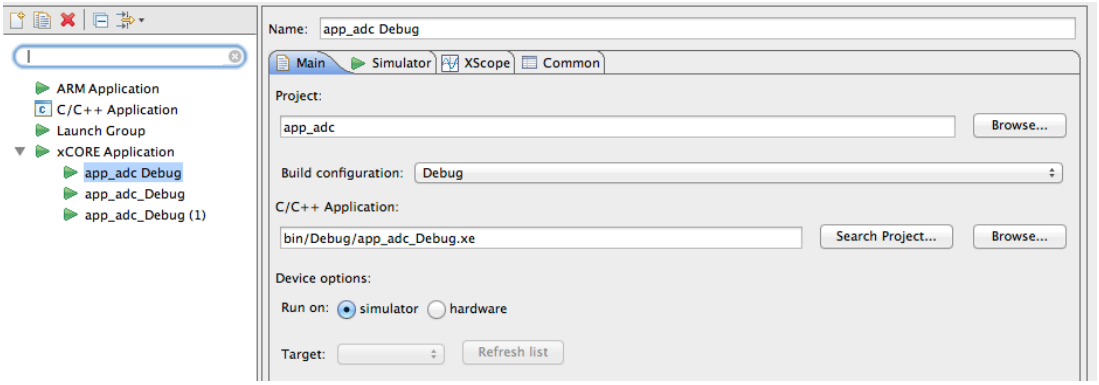

Figure 4: Screen capture showing Run Configuration for simulator

3. Click on the *Simulator* tab and select the **Enable Signal Tracing** checkbox.

4. Check the **Pins** checkbox in the *System Trace options (per device)*.

5. Click the **Add** button, check the **Ports** checkbox and click **Apply** (Figure 5).

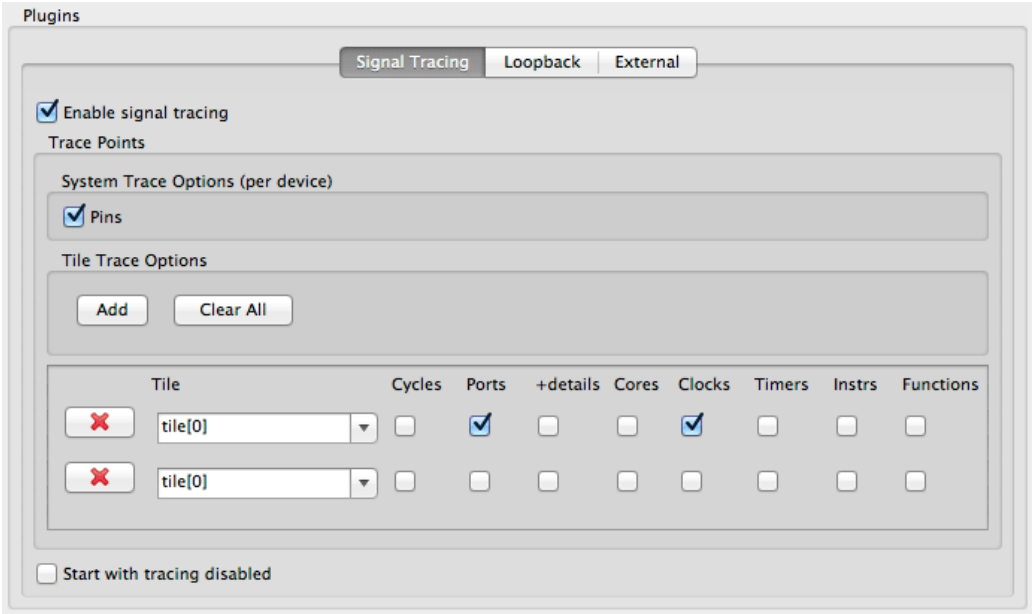

Figure 5: Screen capture showing simulator tab

6. Select the *Loopback* tab and check the **Enable Pin Connections** checkbox.

7. Click the **Add** button and set loopback connections for *Port 1J* to *Port 1G*.

8. Create a loopback for Port 1K to Port 1H, Port 1L to 1C, and Port 1A to 1M (Figure 6).

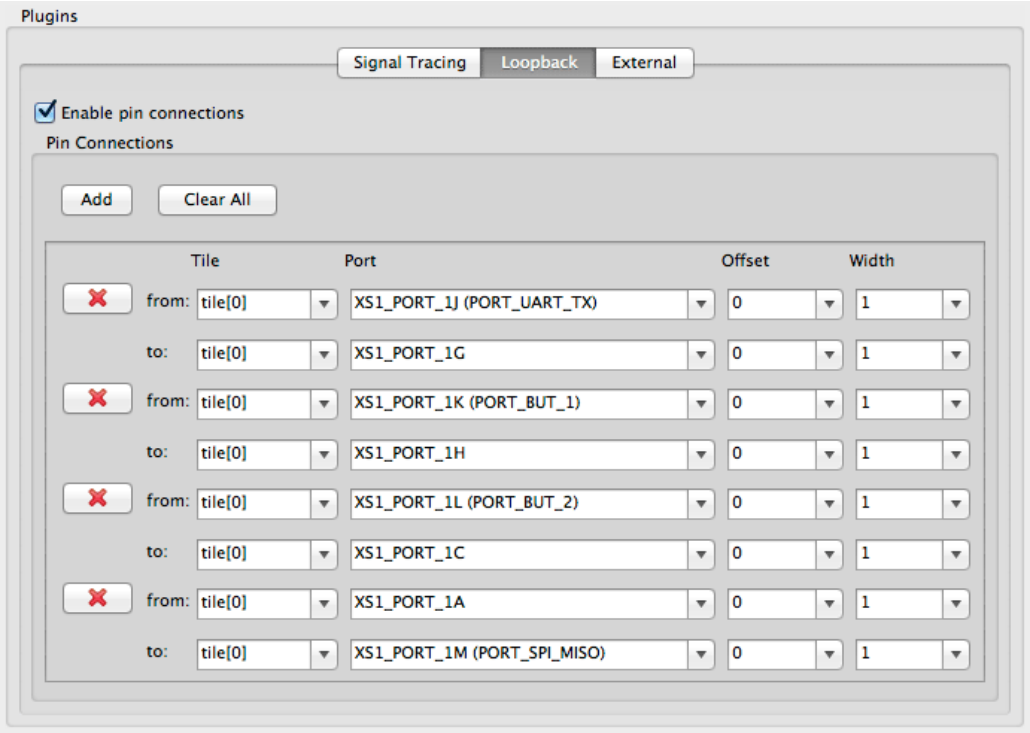

Figure 6: Screen capture showing loopback settings

9. Click **Run** to run the demo using simulator. Once the demo starts running, you will see the application core displaying the Channel A and Channel B values alternatively on the console.

3.1 Viewing the simulator

After the simulator has been running for 15 seconds, click the **STOP** button in the Console. The simulator generates a VCD file in the project. -0.5em

1. Double-click on the VCD file to open the waveform on top of the *Console* view and the *Signals** panel on top of the *Project Explorer*.

2. Click on **Ports** in the *Signals* view, open the *XS1_PORT_1A* folder.

3. Right-click on **tile[0]_XS1_PORT_1A** and select the **Display signal(s)** option. This adds the Port 1A into the waveform view (Figure 7).

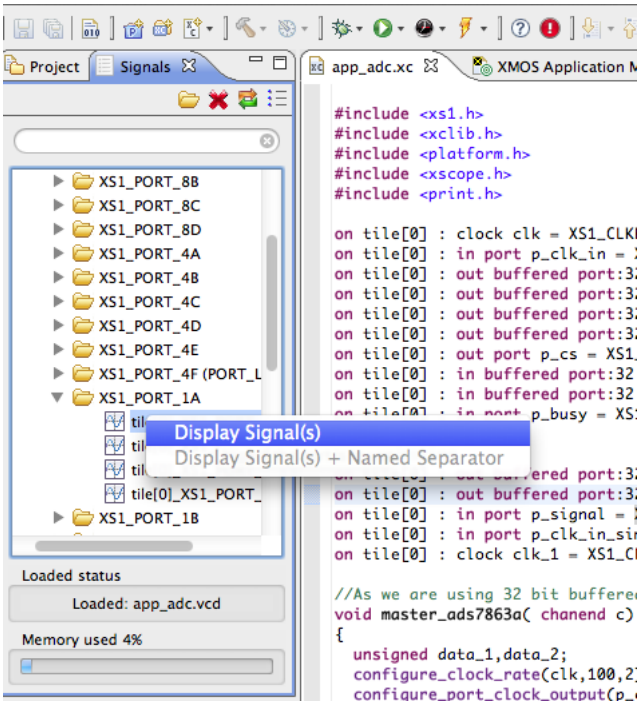

Figure 7: Screen capture showing signals

4. Use the same procedure to add all the ports: 1B, 1C, 1D, 1F, 1G, 1H, 1I, 1J, 1K, 1L, 1M.

5. All the signals are displayed in the waveform viewer, as shown in Figure 8:

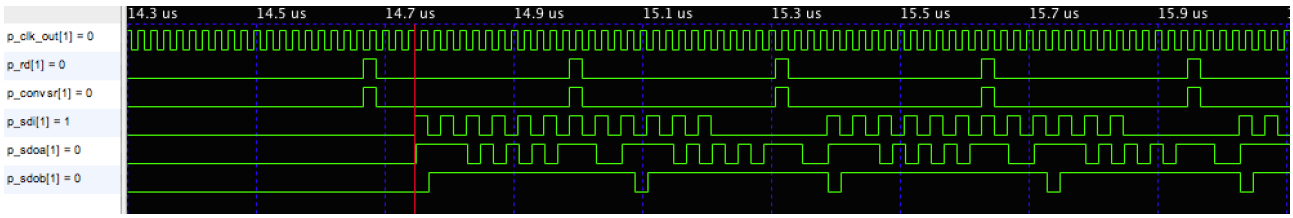

Figure 8: Screen capture showing waveform

3.2 Description of the waveform

- The first signal p_clk_out shows the 50MHz clock signal.
- The p_rd and p_convsr signals show the data these ports is pulled high for every 16 clock pulses.
- The p_sdi carries configuration information for every conversion. So, you will see the data being sent every 16 clock pulses. As we are using 32 bit buffered ports in our application, we send two samples at a time.
- p_sdoa and p_sdob shows the data that is read from from the Channel A and Channel B of ADC after each conversion. We read two samples at a time as we are using 32 bit buffered ports.

4 References

XMOS Tools User Guide

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

XMOS Programming Guide

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

xCORE Architecture Introduction

http://www.xmos.com/published/xcore-architecture

XS1 ports Introduction

http://www.xmos.com/published/xs1-ports-introduction

5 Full source code listing

5.1 Source code for main.xc

```
#include <xs1.h>
#include <xclib.h>
#include <platform.h>

//Ports to interface with ADC
on tile[0] : clock clk = XS1_CLKBLK_1;
on tile[0] : in port p_clk_in = XS1_PORT_1E;
on tile[0] : out buffered port:32 p_clk_out = XS1_PORT_1A;
on tile[0] : out buffered port:32 p_rd = XS1_PORT_1B;
on tile[0] : out buffered port:32 p_convsr = XS1_PORT_1C;
on tile[0] : out buffered port:32 p_sdi = XS1_PORT_1D;
on tile[0] : out port p_cs = XS1_PORT_1F;
on tile[0] : in buffered port:32 p_sdoa = XS1_PORT_1G;
on tile[0] : in buffered port:32 p_sdob = XS1_PORT_1H;
on tile[0] : in port p_busy = XS1_PORT_1I;

//For Slave emulation
on tile[0] : out buffered port:32 p_dummy_1 = XS1_PORT_1J;
on tile[0] : out buffered port:32 p_dummy_2 = XS1_PORT_1K;
on tile[0] : in port p_signal = XS1_PORT_1L;
on tile[0] : in buffered port:32 p_clk_in_sim = XS1_PORT_1M;
on tile[0] : clock clk_1 = XS1_CLKBLK_2;

void master_ads7863a( streaming chanend c)
{
  unsigned data_1,data_2,ts;
  //reference clock is set to 50MHz
  configure_clock_rate(clk,100,2);
  configure_port_clock_output(p_clk_out, clk);
  //Configuring all the ports synchronous to reference clock
  configure_out_port(p_convsr, clk, 0);
  configure_out_port(p_sdi, clk, 0);
  configure_out_port(p_rd, clk, 0);
  configure_in_port(p_sdoa, clk);
  configure_in_port(p_sdob, clk);

  p_rd<:0x0000000;
  p_convsr<:0x00000000;
  p_sdi<:0x00000000;
  set_port_shift_count(p_sdoa,16);
  set_port_shift_count(p_sdoa,16);
  start_clock(clk);
  p_cs<:0;
  while(1)
  {
      p_rd @ (ts+5)<:0x00010001 ;
      p_convsr @ (ts+5) <:0x00010001;
      //write LSB first for two channels at a time
      p_sdi @ (ts+7) <:0x05555550;
      //Read two channels at a time. As we are usign 32 bit buffered ports
      p_sdoa:> data_1;
      p_sdob:> data_2 @ ts;
```

```
      //Send the data read to the application using channels
      c<:data_1;
      c<:data_2;
  }
}

void slave_side_ads7863a_simulation()
{
    unsigned data;
    configure_clock_rate(clk_1,100,2);
    configure_out_port(p_dummy_1, clk_1,0);
    configure_out_port(p_dummy_2, clk_1,0);
    configure_in_port(p_signal, clk_1);
    start_clock(clk_1);

    while(1)
    {
        // wait for conversion start signal from Master
        p_signal:>data ;

        if(data)
        {
            clearbuf(p_dummy_1);
            clearbuf(p_dummy_2);
            //Output Channel A and CHannel B information. Two samples at a time.
            p_dummy_1<: 0x3AAA3AAF ;
            p_dummy_2<: 0xFFFFFFFE;

        }
    }
}

void app( streaming chanend c)
{
    unsigned data;
    while(1)
    {
        //Read Channel A and CHannel B data alternatively.
        c:>data;
    }
}


//main function//
int main()
{
    streaming chan c;
    par
    {
        on tile[0] : master_ads7863a(c);
        on tile[0] : slave_side_ads7863a_simulation();
        on tile[0] : app(c);
    }
}


//End
```