# ADM-XRC-KU1
# Simple (OCP) FPGA Design
# Release: 1.2.0

**Document Revision: 1.3**
**18 July 2022**

| | Head Office | US Office |
|---|---|---|
| Address: | Suite L4A, 160 Dundee Street,<br>Edinburgh, EH11 1DQ, UK | 10822 West Toller Drive, Suite 250<br>Littleton, CO 80127 |
| Telephone: | +44 131 558 2600 | (303) 954 8768 |
| Fax: | +44 131 558 2700 | (866) 820 9956 - toll free |
| email: | sales@alpha-data.com | sales@alpha-data.com |
| website: | http://www.alpha-data.com | http://www.alpha-data.com |

# Table Of Contents



# List of Tables



# List of Figures

# 1 Introduction

**Supported Vivado versions**
This version of the **ADM-XRC-KU1 Simple (OCP) FPGA Design** can be built with Vivado 2018.3 or later.

As of writing, Vivado 2022.1 is the latest release and is recommended. Alpha Data cannot guarantee that this FPGA design will be fully compatible with future releases of Vivado.

This FPGA design demonstrates the use of the **ADM-XRC-KU1-HSOCP** (Host Serial OCP) IP, supplied by Alpha Data, to allow a program running on the host to access registers in the FPGA fabric.

Figure 1 illustrates the ADM-XRC-KU1 within a system when the **ADM-XRC-KU1-HSOCP** IP is used as the target FPGA's host interface:

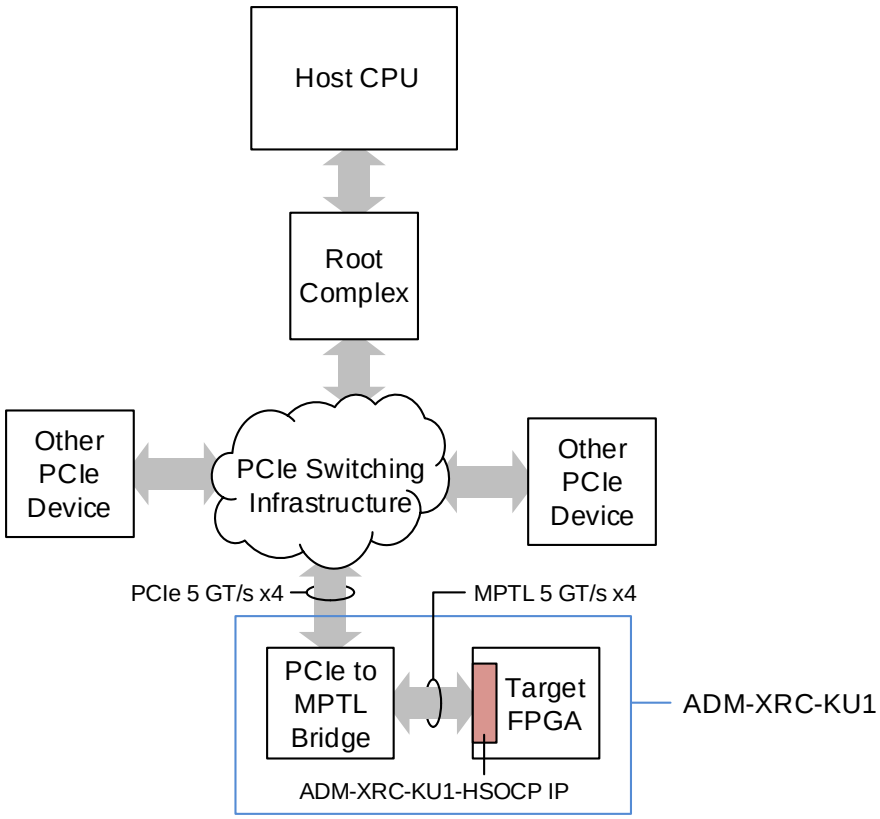


**Figure 1 : The ADM-XRC-KU1 within a system**

Within the ADM-XRC-KU1, the PCIe to MPTL Bridge performs a fixed function, namely to permit the target FPGA to be reconfigured without generating PCI Express errors that are fatal to the system; it is not user-programmable. The target FPGA, on the other hand, is user-programmable and may be reconfigured at will.

## Open Core Protocol

Open Core Protocol (OCP) is the IP interconnect protocol used in many reference designs that were provided by Alpha Data, in **ADM-XRC Gen 3 SDK**, for its reconfigurable computing hardware based on the Xilinx Virtex-6 & 7 Series FPGA families. In some respects, OCP can be regarded as a competitor to the AXI4 standard. Because Xilinx have invested heavily into the AXI4 standard, Alpha Data now recommends the AXI4 protocol for IP interconnection, because this allows the AXI4 IP library in Vivado to be utilized.

To support customers who have invested development effort in FPGA designs using OCP, Alpha Data provides

this FPGA design to demonstrate use of the **ADM-XRC-KU1-HSOCP** IP. In terms of its role in an FPGA design, this IP essentially replaces the **MPTL core** from the **ADM-XRC Gen 3 SDK**. The MPTL cores were provided in the form of a **.ngc** file (a proprietary netlist format used by the Xilinx ISE tools), whereas ADM-XRC-KU1-HSOCP is provided in the form of an IP-XACT definition readable by Vivado. This means that the ADM-XRC-KU1-HSOCP IP is fully compatible with the workflows supported by Vivado.

### ADM-XRC-KU1-HSOCP IP, a replacement for the MPTL core

The ADM-XRC-KU1-HSOCP (Host Serial OCP) IP has several interfaces, including:

- The MPTL (Multiplexed Packet Transport Link) interface; this is an off-chip interface to the Bridge FPGA, implemented using MGTs (Multi-Gigabit Transceivers).
- The Direct Slave interface; this is an OCP master interface through which reads and writes originating on the host CPU can be performed. It is named "Direct Slave" because, from the point of view of the host CPU, the target FPGA is a slave.

  This interface is appropriate for random access, by the host CPU, to registers implemented in the target FPGA.
- Two DMA channels, which are OCP slave interfaces through which reads and writes originating within the PCIe to MPTL Bridge are performed. The DMA channels are not used in this FPGA design, so are not discussed further in this document.

Tcl scripts are provided for generating Vivado projects for the FPGA design, and for building the FPGA design. Refer to Section 3 for details. Using the Simple (OCP) FPGA design in hardware is described in Section 6.

## 1.1 Structure of this package

The files and folders making up the Simple (OCP) FPGA Design are organized as in Figure 2 below:

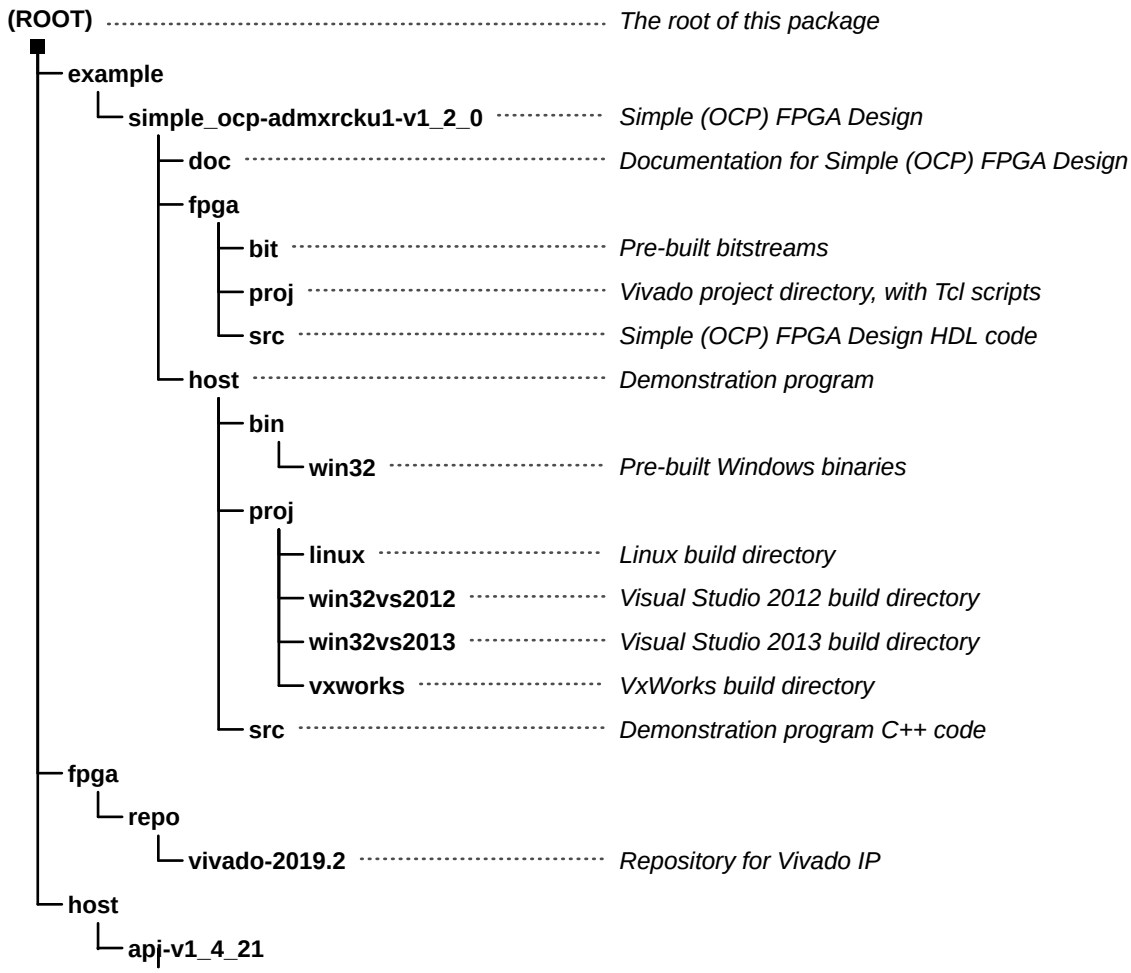


**Figure 2 : Structure of this package**

The root of this package, i.e. the directory which forms the root of the tree of directories and files making up this package, is referred to in the remainder of this document as **(ROOT)**.

The base directory of the FPGA design, i.e. **(ROOT)/example/simple_ocp-admxrcku1-v1_2_0** is referred to in the remainder of this document as **(DESIGN)**.

# 2 Design description

The top level of the FPGA design is implemented by **simple_ocp.vhd**. Figure 3 shows its structure:

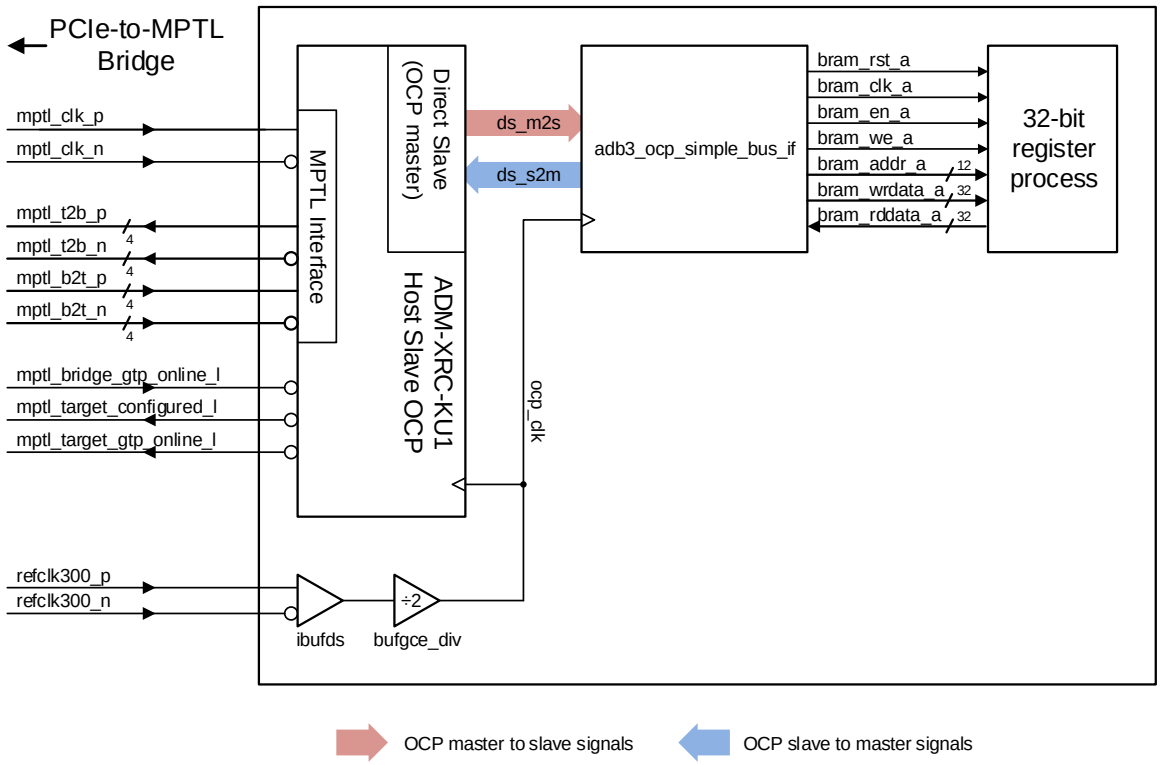


**Figure 3 : Block diagram of FPGA Design**

With reference to figure Figure 3 above, the host CPU (not shown in figure) issues PCI Express read and write transactions which eventually propagate, via the Bridge FPGA of the ADM-XRC-KU1, to the **MPTL interface**. The MPTL interface deserialises them and forwards them to the **Direct Slave** interface, now in the form of OCP transactions.

The OCP transactions are converted into reads and writes on a BlockRAM-style parallel bus. The parallel bus is terminated by a process which implements a single 32-bit register, which returns the last value written, but nibble-reversed, when read back. In other words, the value read back is constructed from the last value written as in Figure 4 below:

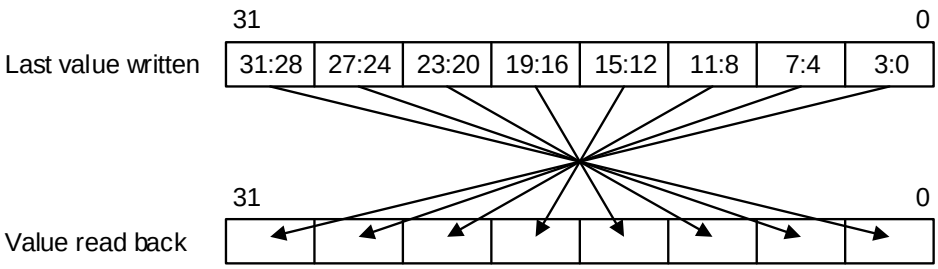


**Figure 4 : Nibble reversal**

The constraints required by the FPGA design are located in **.xdc** files in the **(DESIGN)/fpga/src/** folder. The constraints are split over several files according to function and are listed in Table 1:

| Constraint File | Description |
|---|---|
| bitstream.xdc | Bitstream generation constraints |
| mptl-admxrcku1.xdc | MPTL interface constraints |
| refclk300.xdc | 300 MHz reference clock constraints |
| simple_ocp.xdc | FPGA design general constraints |

**Table 1 : Design Constraints**

## 2.1 Testbench

The testbench is implemented by **tb_simple_ocp.vhd**. Figure 5 shows its structure:

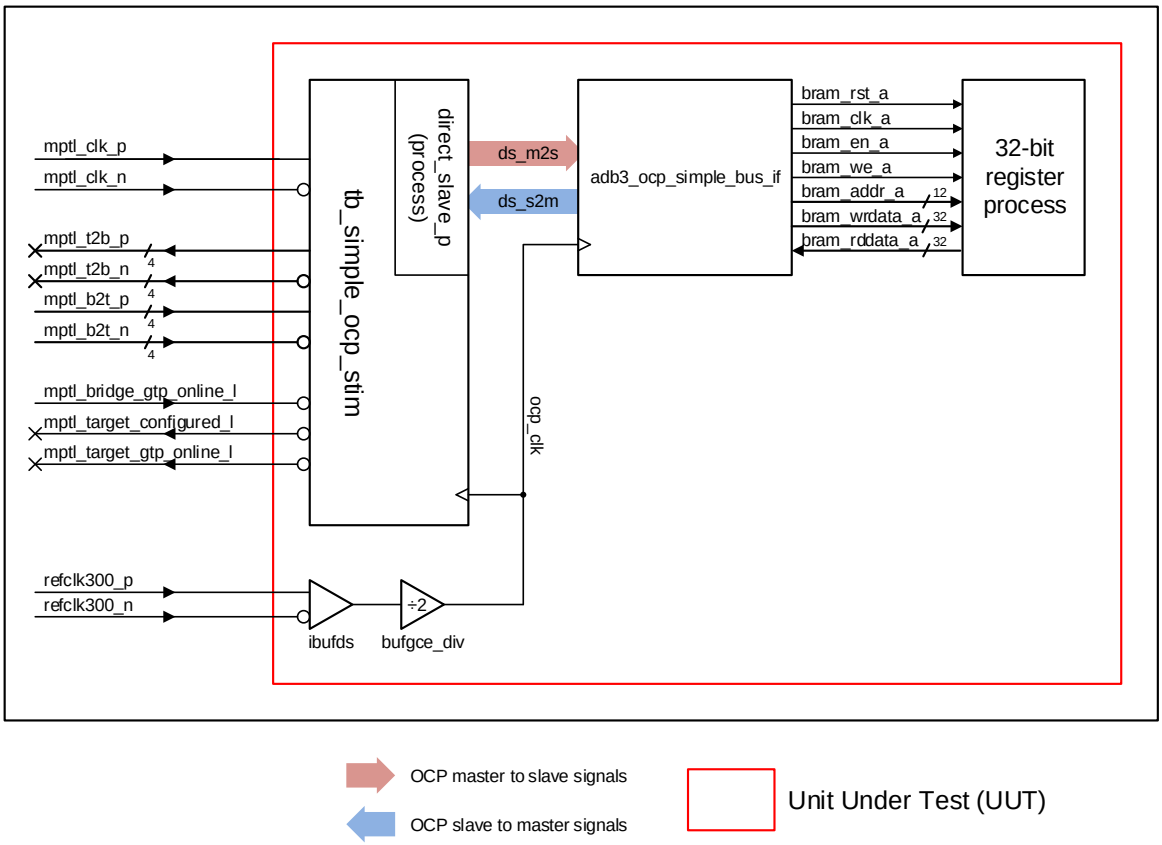


**Figure 5 : Block diagram of Testbench**

The approach taken for simulation is to replace the instance of **ADM-XRC-KU1-HSOCP**, which serves as the host interface, with a behavioral model (**tb_simple_ocp_stim**), whose **direct_slave_p** process stimulates the **ds_ocp_*** channel. It generates OCP transactions that emulate what the host CPU might produce. The MPTL interface is largely ignored by the behavioral model.

The project generation scripts create a simulation source set **sim_ocp** in Vivado which can be used to simulate the design behaviorally, using XSIM or the third-party simulators supported by Vivado. A successful simulation run produces output on the simulator's console of the form (timestamp lines abbreviated for clarity):

```
Note: MPTL now online
Time: 50 ns  Iteration: 2  Process: ...
Note: Wrote simple DATA 4 bytes 0xCAFEFACE with enable 0b1111 to byte address 0x00
00000000000000
Time: 162500 ps  Iteration: 0  Process: ...
Note: Read simple DATA 4 bytes 0xECAFEFAC from byte address 0x0000000000000000
```

```
Time: 232500 ps  Iteration: 0  Process: ...
Note: Write & read back register test completed: PASSED.
Time: 232500 ps  Iteration: 0  Process: ...
Failure: Simulation of design Simple (OCP) completed: PASSED.
Time: 232500 ps  Iteration: 0  Process: ...
```

# 3 Building the Simple (OCP) FPGA Design

Tcl scripts to create the Vivado projects for the various configurations of the FPGA design are found in the **(DESIGN)/fpga/proj/** directory. These can be **source**d within the Vivado GUI, or **source**d by Vivado in batch mode. The available Tcl scripts are listed in Table 2:

| Configuration | Project creation script in **(DESIGN)/fpga/proj/** |
|---|---|
| ADM-XRC-KU1 with KU060-2I | mkxpr-simple_ocp-ku060_2i.tcl |
| ADM-XRC-KU1 with KU115-2I | mkxpr-simple_ocp-ku115_2i.tcl |

**Table 2 : Project creation scripts by configuration**

To generate a project, start a shell or command prompt, and issue a command of the following form:

```
cd /path/to/fpga/proj
vivado -mode batch -source <.tcl script>
```

(Windows users should use backslashes in the **cd** command, rather than forward slashes.)

For example, to generate a Vivado project for an ADM-XRC-KU1 with KU115-2I fitted, (the 2nd configuration in Table 2 above), invoke Vivado as follows:

```
cd /path/to/fpga/proj
vivado -mode batch -source mkxpr-simple_ocp-ku115_2i.tcl
```

(Windows users should use backslashes in the **cd** command, rather than forward slashes.)

After one or more projects have been generated using the scripts listed in Table 2, they can be opened in the Vivado GUI.

Scripts are also provided in the same directory to fully rebuild a Vivado project or all Vivado projects via the shell or command prompt. These are named similarly to the **mkxpr** scripts, except that the prefix is **rebuild**. For example, to rebuild the Vivado project for an ADM-XRC-KU1 with a KU115-2I fitted, invoke Vivado as follows:

```
cd /path/to/fpga/proj
vivado -mode batch -source rebuild-simple_ocp-ku115_2i.tcl
```

(Windows users should use backslashes in the **cd** command, rather than forward slashes.)

> **Note**
> The pre-built bitstreams, under **(DESIGN)/fpga/bit/<configuration>**, are **not** overwritten when the FPGA design is built.

# 4 Demonstration program

The source code of the demonstration program **simple_ocp** is located in **(DESIGN)/host/src/** and consists of three files:

- **cmdline.cpp**

  This file contains the **main** entry point and code for parsing command-line arguments, and nothing in this file is directly related to the FPGA design. It makes use of the **CExAppCmdLineArgs** class, which is provided by the example application framework code in **(ROOT)/host/app_framework-v1_4_0/**.

  Note that when building the demonstration program for VxWorks, this source file is omitted because a VxWorks downloadable kernel module does not have a traditional main()-style entry point.

- **simple_ocp.cpp**

  This file contains code that drives the target FPGA logic, writing values to the nibble-reversal register and reading them back.

- **simple_ocp.h**

  This file defines the interface to the code in **simple_ocp.cpp**, and is used by **cmdline.cpp**.

  Note that in VxWorks, the functions whose prototypes are defined in this file can be called from the VxWorks kernel shell.

The demonstration program works as follows:

1. It opens an ADMXRC3 device either by index or serial number, depending on arguments passed on the command line.
2. It configures the target FPGA with the appropriate pre-built **.bit** file from the **(DESIGN)/fpga/bit/** directory, using the API function **ADMXRC3_ConfigureFromFile**.
3. It maps the nibble-reversal register in the target FPGA into the process' virtual address space, using the API function **ADMXRC3_MapWindow**. This returns as pointer which can be dereferenced to access the nibble-reversal register.
4. It enters a loop:
   (a) It waits for the user to enter a hexadecimal value on the **stdin** stream. If end-of-file occurs or the value entered is 0, the loop terminates.
   (b) It writes the value just entered, as a 32-bit value, to the nibble-reversal register in the target FPGA, using the pointer obtained in step above 3.
   (c) It reads back the nibble-reversal register and displays the value as a hexadecimal number.
5. Before exiting, the program cleans up by unmapping the nibble-reversal register from the process' virtual address space and closing the handle to the ADMXRC3 device.

# 5 Building the demonstration program

## 5.1 Building in Linux

A **Makefile** for GNUMake is provided for building the demonstration program, **simple_ocp**. The GNU C++ toolchain and associated C and C++ development packages must be installed in the system that is used to build **simple_ocp**.

To build **simple_ocp**, follow this procedure:

1. Start a shell and change directory to **(DESIGN)/host/proj/linux**.
2. Issue the following command:

   ```
   make
   ```

Assuming that building is successful, the executable is **(DESIGN)/host/proj/linux/simple_ocp**.

The above procedure builds **simple_ocp** natively, i.e. for the architecture that the GNU toolchain on the build system targets by default. There are two variables that may be passed on the **make** command-line or set in the environment in order to change the way building is performed:

- **BIARCH**

  For most 64-bit Linux distributions, it is possible to build both a native (64-bit) executable and a 32-bit executable. To do this, set **BIARCH** variable to yes on the **make** command-line. For example:

  ```
  make BIARCH=yes
  ```

  Assuming that building is successful, the executables produced are **simple_ocp** (native 64-bit) and **simple32** (32-bit).

- **CROSS_COMPILE**

  To build using a cross-compiler, set the **CROSS_COMPILE** environment variable to the prefix of the toolchain binaries, ensuring that the toolchain is in the **PATH**. For example

  ```
  export PATH=/path/to/toolchain:$PATH
  export CROSS_COMPILE=arm-none-linux-gnueabi-
  make
  ```

- **SYSROOT**

  Generally used only when cross-compiling, the value of **SYSROOT** points to the target system's root filesystem. This may be required if the toolchain used for cross-compiling does not have the required defaults for paths to system header files and libraries directories. For example:

  ```
  export PATH=/path/to/toolchain:$PATH
  export CROSS_COMPILE=arm-none-linux-gnueabi-
  make SYSROOT=/path/to/arm-rootfs
  ```

## 5.2 Building in Windows

Solutions for Microsoft Visual Studio 2012 & 2013 are provided for building the demonstration program, **simple_ocp.exe**.

To build **simple_ocp.exe** for a particular configuration-platform combination, follow this procedure:

1. If using Microsoft Visual Studio 2012, open the solution **(DESIGN)/host/proj/win32vs2012/simple_ocp.sln**.

   If using Microsoft Visual Studio 2013, open the solution **(DESIGN)/host/proj/win32vs2013/simple_ocp.sln**.
2. From the **Standard** toolbar, which is visible by default in Microsoft Visual Studio, select the configuration and platform of interest; for example **Release**, **x64**.
3. From the main menu, select **BUILD** -> **Rebuild Solution**.

Alternatively, follow this procedure to build all available configuration-platform combinations of **simple_ocp.exe**:

1. If using Microsoft Visual Studio 2012, open the solution **(DESIGN)/host/proj/win32vs2012/simple_ocp.sln**.

   If using Microsoft Visual Studio 2013, open the solution **(DESIGN)/host/proj/win32vs2013/simple_ocp.sln**.
2. From the main menu, select **BUILD** -> **Batch Build...**
3. In the **Batch Build** dialog, click **Select All** and then **Rebuild**.

Once built, the executable files for **simple_ocp.exe** are located as follows, according to Visual Studio version, configuration and platform:

| Visual Studio | Configur-ation | Platform | Executable location |
|---|---|---|---|
| 2012 | Debug | Win32 | **(DESIGN)/host/proj/win32vs2012/simple_ocp/Debug/** |
| 2012 | Debug | x64 | **(DESIGN)/host/proj/win32vs2012/simple_ocp/Debug64/** |
| 2012 | Release | Win32 | **(DESIGN)/host/proj/win32vs2012/simple_ocp/Release/** |
| 2012 | Release | x64 | **(DESIGN)/host/proj/win32vs2012/simple_ocp/Release64/** |
| 2013 | Debug | Win32 | **(DESIGN)/host/proj/win32vs2013/simple_ocp/Debug/** |
| 2013 | Debug | x64 | **(DESIGN)/host/proj/win32vs2013/simple_ocp/Debug64/** |
| 2013 | Release | Win32 | **(DESIGN)/host/proj/win32vs2013/simple_ocp/Release/** |
| 2013 | Release | x64 | **(DESIGN)/host/proj/win32vs2013/simple_ocp/Release64/** |

**Table 3 : Location of simple_ocp.exe**

## 5.3 Building for VxWorks

A **Makefile** is provided for building a downloadable kernel module, **admxrc3SimpleOcp.out**, which has entry points that may be called from the VxWorks shell.

To invoke the **Makefile**, follow these steps:

1. Start a VxWorks Development Shell. This can be started from within Workbench or from the Start Menu if running in Windows.
2. In the shell, change directory to **(DESIGN)/host/proj/vxworks**.
3. Issue the **make** command, specifying the CPU architecture, toolchain and other build options. For example:

   ```
   make CPU=NEHALEM TOOL=icc VXBUILD="LP64 SMP" clean default
   ```

   The above command builds **admxrc3SimpleOcp.out** for 64-bit SMP Nehalem architecture using the Intel toolchain.

Assuming that the **make** command is successful, the build product is **admxrc3SimpleOcp.out**, which can be downloaded to the target system.

For a more detailed discussion of how to invoke the **Makefile**, refer to Appendix C.

# 6 Using the Simple (OCP) FPGA Design

## 6.1 Using the FPGA Design with a Linux host

The demonstration program, **simple_ocp**, runs on the host system's CPU and writes values entered by the user into the nibble-reversal register of the target FPGA. The nibble-reversed values are read back and displayed. Before doing so, please ensure that your environment meets the following requirements:

- An ADM-XRC-KU1 is plugged into an XMC slot in the test machine and SW1-3 (Bridge Bypass mode) is OFF.
- ADB3 Driver 1.4.17 or later is installed in the test machine.
- You have built the demonstration program as detailed in Section 5.1.
- You are logged in as a user that is capable of executing programs as **root** using **sudo**.

### Start the ADB3 Driver

If the ADB3 Driver is not already started, start it using the command:

```
sudo modprobe adb3
```

### Run the demonstration program

To run the **simple_ocp** program with default arguments, issue the following commands in a shell:

```
cd (DESIGN)/host/proj/linux
sudo ./simple_ocp
```

This should yield output initially as follows, prompting you to enter values:

```
INFO:  =========================
INFO:  Enter values for I/O
INFO:  (CTRL-D or enter 0 to exit)
INFO:  =========================
```

Once the above prompt has appeared, it is possible to enter 32-bit values. Each one is written to the FPGA where it is nibble-reversed. The program reads back the nibble-reversed values and displays them. For example, after entering the hexadecimal values **1234abcd**, **deadbeef** & **cafeface**, the output looks like this:

```
INFO:  =========================
INFO:  Enter values for I/O
INFO:  (CTRL-D or enter 0 to exit)
INFO:  =========================
1234abcd
INFO:  OUT = 0x1234ABCD, IN = 0xDCBA4321
deadbeef
INFO:  OUT = 0xDEADBEEF, IN = 0xFEEBDAED
cafeface
INFO:  OUT = 0xCAFEFACE, IN = 0xECAFEFAC
```

When finished, either press CTRL-D or enter 0 to exit the program.

## 6.2 Using the FPGA Design with a Windows host

The demonstration program, **simple_ocp**, runs on the host system's CPU and writes values entered by the user into the nibble-reversal register of the target FPGA. The nibble-reversed values are read back and displayed. Before running it, please ensure that your environment meets the following requirements:

- An ADM-XRC-KU1 is plugged into an XMC slot in the test machine and SW1-3 (Bridge Bypass mode) is OFF.

- ADB3 Driver 1.4.17 or later is installed in the test machine.
- You are either:
  - Logged in as a user with Administrator privileges in a system without User Account Control (UAC) or where UAC is disabled, and have started a Windows Command Prompt (which will be elevated).
  - Logged in as a user with Administrator privileges in a system with User Account Control, and have started a Windows Command Prompt using "Run as administrator".

### Run the demonstration program

To run the **simple_ocp.exe** program with default arguments, issue the following commands in the Windows Command Prompt:

```
cd (DESIGN)\host\bin\win32\x86
simple_ocp
```

This should yield output initially as follows, prompting you to enter values:

```
INFO:  ========================
INFO:  Enter values for I/O
INFO:  (CTRL-Z or enter 0 to exit)
INFO:  ========================
```

Once the above prompt has appeared, it is possible to enter 32-bit values. Each one is written to the FPGA, where it is nibble-reversed. The program reads back the nibble-reversed values and displays them. For example, after entering the hexadecimal values **1234abcd**, **deadbeef** & **cafeface**, the output looks like this:

```
INFO:  ========================
INFO:  Enter values for I/O
INFO:  (CTRL-Z or enter 0 to exit)
INFO:  ========================
1234abcd
INFO:  OUT = 0x1234ABCD, IN = 0xDCBA4321
deadbeef
INFO:  OUT = 0xDEADBEEF, IN = 0xFEEBDAED
cafeface
INFO:  OUT = 0xCAFEFACE, IN = 0xECAFEFAC
```

When finished, either enter CTRL-Z or enter 0 to exit the program.

## 6.3 Using the FPGA Design in VxWorks

The demonstration program, **simple_ocp**, runs on the host system's CPU and writes values entered by the user into the nibble-reversal register of the target FPGA. The nibble-reversed values are read back and displayed. Before running it, please ensure that your environment meets the following requirements:

- An ADM-XRC-KU1 is plugged into an XMC slot in the VxWorks target machine and SW1-3 (Bridge Bypass mode) is OFF.
- ADB3 Driver 1.4.17 or later has been built and is running on the VxWorks target machine. This can be done by one of two methods:
  (a) By downloading ADB3 Driver, as a set of downloadable kernel modules (DKMs), to the VxWorks target machine, after booting. For this method, please refer to the release notes for **ADB3 Driver for VxWorks**.
  (b) By building ADB3 Driver Component into the VxWorks kernel so that it is automatically started when the kernel boots. For this method, please refer to the release notes for **ADB3 Driver Component for VxWorks**.
- You have built the demonstration program as detailed in Section 5.3.
- You have access to the kernel shell on the VxWorks target machine, either using a serial connection or using telnet.

### Download the demonstration program to the VxWorks target machine

Assuming that you have built it as described in Section 5.3, the DKM for the demonstration program must first be downloaded to the VxWorks target machine. This can be done by a shell command such as:

```
-> ld <HOST:(DESIGN)/host/proj/vxworks/admxrc3SimpleOcp.out
value = -140737478381552 = 0xffff800000983010
```

where *HOST* is the VxWorks host.

> **Undefined symbols when loading the DKM**
> If the **ld** command fails due to undefined symbols, the most likely cause is that the ADB3 Driver has not been correctly downloaded to the VxWorks target system.

### Run the demonstration program

Once the DKM for the demonstration program is resident in the VxWorks target system, it is possible to run it. The basic form of shell command that runs the program uses the **admxrc3SimpleIndex** entry point in the DKM, and requires the path to the **(DESIGN)/fpga/bit/** directory to be the first argument:

```
-> admxrc3SimpleIndex "HOST:(DESIGN)/simple_ocp-admxrcku1-v1_2_0/fpga/bit"
```

where *HOST* is the VxWorks host.

Starting the program should yield output initially as follows, prompting you to enter values:

```
INFO:  ========================
INFO:  Enter values for I/O
INFO:  (CTRL-Z or enter 0 to exit)
INFO:  ========================
```

Once the above prompt has appeared, it is possible to enter 32-bit values. Each one is written to the FPGA where it is nibble-reversed. The program reads back the nibble-reversed values and displays them. For example, after entering the hexadecimal values **1234abcd**, **deadbeef** & **cafeface**, the output looks like this:

```
INFO:  ========================
INFO:  Enter values for I/O
INFO:  (CTRL-D or enter 0 to exit)
INFO:  ========================
1234abcd
INFO:  OUT = 0x1234ABCD, IN = 0xDCBA4321
deadbeef
INFO:  OUT = 0xDEADBEEF, IN = 0xFEEBDAED
cafeface
INFO:  OUT = 0xCAFEFACE, IN = 0xECAFEFAC
```

When finished, enter 0 to exit the program.

# Appendix A: Demonstration program options in Linux & Windows

The demonstration program, **simple_ocp[.exe]** may be invoked with a number of options and positional arguments:

```
simple_ocp [option ...]
```

Options begin with '-'. If an option requires a value, it may be specified in one or two forms: **-option=*<value>*** or **-option *<value>***. The available options are:

- **-h**, **-help**, **-?**

  This option displays a brief help message.

- **-bitfile *</path/to/file.bit>***

  This option specifies the **.bit** file to be used to configure the FPGA, and takes precedence over the **-bitpath** option. In other words, if **-bitfile** is passed on the command line, the value of the **-bitpath** option is ignored.

  It may be necessary to use the **-bitfile** option in order to directly specify the **.bit** file when the program does not compute the correct path to the **.bit** file; for example, when the FPGA fitted to the board in use is of a speed grade other than **-2** or a temperature grade other than **E**.

- **-bitpath *</path/to/bit_directory>***

  This option specifies the path of **bit** directory whose subdirectories represent configurations of the FPGA design and contain pre-built **simple_ocp.bit** files. Its default value is a relative path which corresponds to **(DESIGN)/fpga/bit**.

  Assuming that the **-bitfile** option is not passed on the command line, the filename of the **.bit** file to be used to configure the FPGA is computed as follows:

  **<-bitpath value>**/simple_ocp-**<device>_<speed><tempgrade>**[_**<step>**]/simple_ocp.bit

  where "device", "speed", "tempgrade" and "step" are all obtained via the **ADMXRC3_GetFPGAInfo** function of the ADMXRC3 API. The "step" value is generally empty for a board fitted with a production silicon FPGA, and in that case is omitted from the **.bit** file path.

  For example, for an ADM-XRC-KU1 fitted with a KU115-2I device, the computed **.bit** file partially evaluates to:

  *<-bitpath value>*/simple_ocp-ku115_2i/simple_ocp.bit

  Note: If the **-bitfile** option is passed on the command line, this **-bitpath** option's value is completely ignored.

- **-index *<index>***

  This option specifies which reconfigurable computing device is to be used for the test. Zero corresponds to the first reconfigurable computing device in the system, as enumerated by the operating system. 1 corresponds to the second device, and so on.

  If omitted, the value is 0. This option cannot be specified along with the **-sn** option (see below).

  Examples:

  - **-index 0**

    Use the first reconfigurable computing device in the system.

  - **-index 10**

    Use the 11th reconfigurable computing device in the system.

  - **-index 0x2**

    Use the third reconfigurable computing device in the system.

- **-sn *<serial number>***

This option specifies the serial number of the reconfigurable computing device that is to be used for the test.

If omitted, the device used is chosen according to the **-index** option (see above). This option cannot be specified along with the **-index** option (see above).

Examples:

- **-sn 159**

  Use the reconfigurable computing device with serial number 159.

- **-sn 0x5555**

  Use the reconfigurable computing device with serial number 0x5555 (21845).

# Appendix B: Demonstration program entry points in VxWorks

The demonstration program can be invoked via two entry points in the **admxrc3SimpleOcp.out** DKM. These entry points are defined by the header file, **(DESIGN)/host/src/simple_ocp.h**, as follows:

```
int
admxrc3SimpleIndex(
  const TCHAR* pBitPath,
  const TCHAR* pBitFile,
  unsigned int index);

int
admxrc3SimpleSN(
  const TCHAR* pBitPath,
  const TCHAR* pBitFile,
  uint32_t     serialNumber);
```

**Use of TCHAR**
The demonstration program is portable between Linux, Windows and VxWorks. For this reason, **TCHAR** is used as the character data type, and when building for VxWorks, **TCHAR** is aliased to **char**.

- **admxrc3SimpleIndex**

  This entry point is for running the demonstration program on an ADM-XRC-KU1 with a particular zero-based **index**. If there is only one card in the system, its index is always 0.

- **admxrc3SimpleSN**

  This entry point is for running the demonstration program on an ADM-XRC-KU1 with a particular **serial number**.

The parameters are as follows:

- **pBitPath**

  If non-NULL, the **pBitPath** argument specifies the directory on the host filesystem where the pre-built **.bit** files are located. It is used as the prefix for constructing a full path to the **.bit** file to be used to configure the target FPGA, which is performed as follows (where + represents string concatenation):

  **pBitPath** + "/simple_ocp-**<device>_<speed><tempgrade>**[**_<step>**]/simple_ocp.bit"

  where "device", "speed", "tempgrade" and "step" are all obtained via the **ADMXRC3_GetFPGAInfo** function of the ADMXRC3 API. The "step" value is generally empty for a board fitted with a production silicon FPGA, and in that case is omitted from the **.bit** file path.

  For example, for an ADM-XRC-KU1 fitted with a KU115-2I device, the full path of the **.bit** file is constructed as:

  **pBitPath** + "/simple_ocp-ku115_2i/simple_ocp.bit"

- **pBitFile**

  If non-NULL, the **pBitFile** argument directly specifies the **.bit** file to use to configure the FPGA. Its value overrides whatever **.bit** file path was constructed from **pBitPath**.

  If **pBitPath** is NULL, **pBitFile** must be given a non-NULL value so that the program knows what **.bit** file to use.

- **index**

  In the **admxrc3SimpleIndex** entry point, this parameter specifies the zero-based index of the reconfigurable computing card to use. If there is only one reconfigurable computing card in the system, its index is always zero. When there are more than one, indices are assigned by the system, generally according to the order in which they are discovered.

- **serialNumber**

  In the **admxrc3SimpleSN** entry point, this parameter specifies the serial number of the reconfigurable computing card to use.

If **pBitFile** is not NULL, it overrides any value passed for **pBitPath**. Table 4 summarizes the interaction of **pBitPath** and **pBitFile**:

| pBitPath | pBitFile | Behavior |
|---|---|---|
| NULL | NULL | Illegal; the program does not know what **.bit** file to use. |
| non-NULL | NULL | The program constructs the full path of the **.bit** file from **pBitPath** and information obtained via **ADMXRC3_GetFPGAInfo**. |
| N/A | non-NULL | The program uses **pBitFile** as the full path of the **.bit** file |

**Table 4 : Interaction of pBitPath and pBitFile**

# Appendix C: Makefile variables in VxWorks

The **Makefile** for building the downloadable kernel module (DKM) **admxrc3SimpleOcp.out** in VxWorks can be invoked with a number of variables for controlling how the build is performed. The general form is:

```
make [CPU=<arch>] [TOOL=<tool>] [VXBUILD="[option] ..."] [target ...]
```

The available build targets for **make** are:

- **clean**

  This deletes all build products and intermediate files. When rebuilding with different values for **CPU**, **TOOL** etc. with respect to the previous build, first perform a clean.

- **default**

  This builds the product **admxrc3SimpleOcp.out** according to the values for **CPU**, **TOOL** etc.

To perform a full rebuild, use both **clean** and **default** together in the same command, in that order.

The variables that may be passed on the **make** command-line are:

- **CPU=*<arch>***

  Here, ***<arch>*** is the CPU architecture of the target system; for example **PPC604**, **NEHALEM**, **ARMARCH4** etc.

  If this variable is omitted, it defaults to **PPC604**.

- **TOOL=*<tool>***

  Here, ***<tool>*** is the toolchain that is to be used to build the DKM and, as of VxWorks 6.9, can be **diab**, **gnu** or **icc**.

  If this variable is omitted, it defaults to **gnu**.

- **VXBUILD="*[option]* ..."**

  Here the properties of the kernel of the target system must be specified. Including **LP64** means that the kernel of the target system is a 64-bit kernel. Including **SMP** means that the kernel of the target system is symmetric multiprocessing (SMP). Any options that are included should be separated by spaces, with all options together enclosed in quotes. For example, for a 64-bit SMP kernel, use

  ```
  VXBUILD="LP64 SMP"
  ```

  If this variable is omitted, it defaults to "", the result of which depends upon the defaults for the architecture selected by **CPU**. For example, if **CPU** is **PPC604** or **NEHALEM**, omitting **VXBUILD** results in building for a 32-bit uniprocessor kernel.

Hence, to fully rebuild for a 32-bit uniprocessor PowerPC 604 kernel using the GNU toolchain, issue the command

```
make clean default
```

To build for a 64-bit SMP Nehalem kernel using the Intel toolchain, issue the command

```
make CPU=NEHALEM TOOL=icc VXBUILD="LP64 SMP" default
```

# Revision History

| Date | Revision | Nature of change |
|---|---|---|
| 22 June 2016 | 1.0 | Initial version for simple_ocp-admxrcku1-1.0.0. |
| 2 August 2016 | 1.1 | Updated for temperature grade change from E to I. |
| 31 August 2016 | 1.2 | Changed from using api-1.4.17 to api-1.4.18b4. |
| 18 July 2022 | 1.3 | Updated for Vivado 2018.3 to 2022.1. |

Address: Suite L4A, 160 Dundee Street, Edinburgh, EH11 1DQ, UK
Telephone: +44 131 558 2600
Fax: +44 131 558 2700
email: sales@alpha-data.com
website: http://www.alpha-data.com

Address: 10822 West Toller Drive, Suite 250 Littleton, CO 80127
Telephone: (303) 954 8768
Fax: (866) 820 9956 - toll free
email: sales@alpha-data.com
website: http://www.alpha-data.com