ALPHA DATA

ADM-XRC-KU1 DMA Demonstration FPGA Design Release: 1.0.0

Document Revision: 1.0
22 Jun 2016

	Head Office	US Office
Address:	4 West Silvermills Lane, Edinburgh, EH3 5BD, UK	3507 Ringsby Court Suite 105, Denver, CO 80216
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 DMA Demonstration FPGA Design** can be built with Vivado 2015.4 or later.

As of writing, Vivado 2016.2 is the latest release and is recommended. Alpha Data cannot guarantee that this FPGA design will be fully compatible with future releases of Vivado.

Please review Xilinx Quality Alert XCN15040 .

This FPGA design demonstrates the use of the **ADM-XRC-KU1-HSAXI** (Host Slave AXI4) IP, supplied by Alpha Data, to allow a program running on the host to transfer in bulk to and from the target FPGA in an ADM-XRC-KU1 reconfigurable computing card.

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

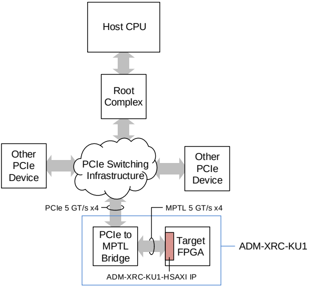


Figure 1 : The ADM-XRC-KU1 within a system

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.

The **ADM-XRC-KU1-HSAXI** (Host Slave AXI4) IP is instantiated in the target FPGA and has several interfaces:

- 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 AXI4 memory-mapped (AXI4-MM) 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 AXI4 memory-mapped (AXI4-MM) slave interfaces through which reads and writes originating within the PCIe to MPTL Bridge are performed.

  These interfaces are appropriate for bulk data transfer between host memory and the target FPGA.

This FPGA design demonstrates instantiating ADM-XRC-KU1-HSAXI so that data can be transferred by the DMA engines (located in the Bridge FPGA) between host memory and BlockRAMs in the target FPGA. In addition, the BlockRAMs can be accessed via the Direct Slave channel so that DMA transfers can be verified for correctness of data.

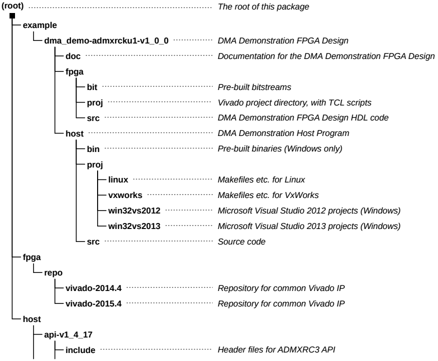
Tcl scripts are provided for generating Vivado projects for the FPGA design, and for building the FPGA design. Refer to Section 3 for details. Using the Simple FPGA design in hardware is described in Section 6.

To exercise the FPGA design, a program that runs on the host is provided. It measures the data transfer performance of a single DMA engine, or multiple DMA engines in aggregate, depending on command-line options. For details of building this program, refer to Section 5.

Using the FPGA design in hardware with the demonstration program is described in Section 6.

1.1 Structure of this package

The files and folders making up the DMA Demonstration FPGA Design are organized as in Figure 2 below:


(root) — The root of this package
example
dma_demo-admxrcku1-v1_0_0 — DMA Demonstration FPGA Design
doc — Documentation for the DMA Demonstration FPGA Design
fpga
bit — Pre-built bitstreams
proj — Vivado project directory, with TCL scripts
src — DMA Demonstration FPGA Design HDL code
host — DMA Demonstration Host Program
bin — Pre-built binaries (Windows only)
proj
linux — Makefiles etc. for Linux
vxworks — Makefiles etc. for VxWorks
win32vs2012 — Microsoft Visual Studio 2012 projects (Windows)
win32vs2013 — Microsoft Visual Studio 2013 projects (Windows)
src — Source code
fpga
repo
vivado-2014.4 — Repository for common Vivado IP
vivado-2015.4 — Repository for common Vivado IP
host
api-v1_4_17
include — Header files for ADMXRC3 API

│ └ **lib** ······························ *Libraries for ADMXRC3 API*
└ **app_framework-v1_1_0** ··············· *Example application framework*

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/dma_demo-admxrcku1-v1_0_0** is referred to in the remainder of this document as **(design)**.

2 Design description

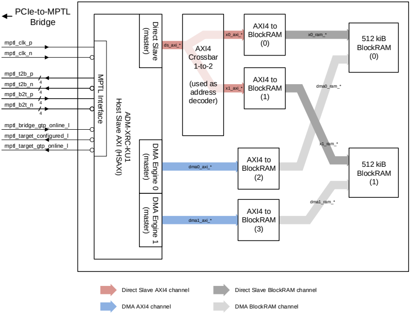


Figure 3 : Block diagram of DMA Demonstration FPGA Design

The FPGA design consists of the following elements:

- An instance of **ADM-XRC-KU1-HSAXI**, which is an IP block supplied by Alpha Data, providing (amongst other things) an MPTL data channel, two DMA channels and a memory mapped Direct Slave interface.
- Two 512 kiB BlockRAMs, one per DMA channel. The BlockRAMs are dual-ported so that they may be simultaneously accessed by both the host and the corresponding DMA channel, although it is the host's responsibility to either cope with or avoid read/write and write/write collisions.
- For each BlockRAM, there are two AXI4 to BlockRAM interfaces. The first allows a particular DMA channel to read and write it, and the second permits the Direct Slave channel to read and write it. Altogether, there are four AXI4 to BlockRAM interfaces in the design.
- An AXI4 crossbar instance for decoding Direct Slave addresses and routing AXI4 commands to the appropriate BlockRAM interface. This splits the Direct Slave AXI4 channel into two channels, according to address. The address decoding scheme is detailed in Table 1.

The Host Slave address map consists of two 512 kiB regions, one per BlockRAM:

Address range	Size	Purpose
0x00000 - 0x0FFFF	512 kiB	Permits the host to read and write BlockRAM 0.
0x80000 - 0xFFFFF	512 kiB	Permits the host to read and write BlockRAM 1.
Others		Reserved; must not be accessed.

Table 1 : Direct Slave AXI4 address map

The DMA channels each see a trivial address map consisting of a single region of 512 kiB. If *N* is the index of a DMA channel, its address map as follows:

Address range	Size	Purpose
0x0 - 0x7FFFF	512 kiB	Permits DMA channel *N* to read and write BlockRAM *N*.
Others		Reserved; must not be accessed.

Table 2 : DMA channel N AXI4 address map

2.1 Testbench

The testbench is implemented by **tb_dma_demo.vhd**. Figure 4 shows its structure:

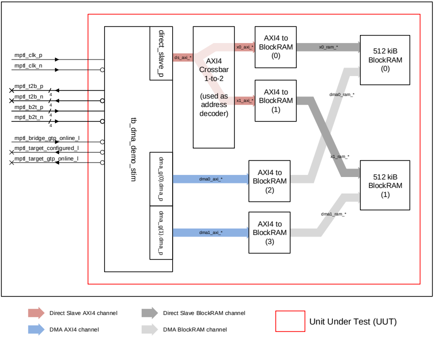


Figure 4 : Block diagram of Testbench

The approach taken for simulation is to replace the instance of **ADM-XRC-KU1-HSAXI**, which serves as the host interface, with a behavioral model (**tb_dma_demo_stim**) that stimulates the **ds_axi_***, **dma0_axi_*** and

dma1_axi_* channels. It generates AXI4-MM transactions that emulate those produced by the host CPU, in the case of the **ds_axi_*** channel, and those produced by the DMA engines in the Bridge FPGA, in the case of the **dma0_axi_*** and **dma1_axi_*** channels. The MPTL interface is largely ignored by the behavioral model.

The project generation scripts create a simulation source set **sim_axi** 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 DS DATA 16 bytes 0x00000003000000020000000100000000 with enable 0b1111
111111111111 to byte address 0x0000000000000000
Time: 107500 ps  Iteration: 0  Process: ...
Note: Wrote DS DATA 16 bytes 0x00000007000000060000000500000004 with enable 0b1111
111111111111 to byte address 0x0000000000000010
Time: 162500 ps  Iteration: 0  Process: ...
... other similar messages ...
Note: Test DS (AXI) completed: PASSED.
Time: 8377500 ps  Iteration: 0  Process: ...
Note: Read DMA0 DATA 512 bytes from byte address 0x0000000000000000
Time: 8597500 ps  Iteration: 0  Process: ...
Note: Read DMA1 DATA 512 bytes from byte address 0x0000000000000000
Time: 8597500 ps  Iteration: 0  Process: ...
... other similar messages ...
Note: Test DMA0 (AXI) completed: PASSED.
Time: 9022500 ps  Iteration: 0  Process: ...
Note: Test DMA1 (AXI) completed: PASSED.
Time: 9022500 ps  Iteration: 0  Process: ...
Failure: Test of design DMA_DEMO (AXI) completed: PASSED.
Time: 9022500 ps  Iteration: 2  Process: ...
```

3 Building the 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 3:

Configuration	Project creation script in **(design)/fpga/proj/**
ADM-XRC-KU1 with KU060-2E	mkxpr-dma_demo-ku060_2e.tcl
ADM-XRC-KU1 with KU115-2E	mkxpr-dma_demo-ku115_2e.tcl

Table 3 : Project creation scripts by configuration

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

```
cd (design)/fpga/proj
vivado -mode batch -source mkxpr-dma_demo-ku060_2e.tcl
```

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

After the project has been created using the script, it can be opened in the Vivado GUI.

A TCL script is also provided in the same directory to fully rebuild the Vivado project via the shell or Command Prompt. This is named similarly to the **mkxpr** script, except that the prefix is **rebuild**. For example, to rebuild the Vivado project, invoke Vivado as follows:

```
cd (design)/fpga/proj
vivado -mode batch -source rebuild-dma_demo-ku060_2e.tcl
```

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

Assuming that building is successful, the newly-built **.bit** file is:

(design)/fpga/proj/<project directory>/dma_demo.runs/impl_1/dma_demo.bit

Note
Pre-built bitstreams, which are found under the directory **(design)/fpga/bit/**, are **not** overwritten when the FPGA design is built.

4 Demonstration program

The demonstration program, **dma_demo** is located in **(design)/host/src/** and consists of three source 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-trunk/**.

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

- **dma_demo.cpp**

  This file contains code that actually drives the FPGA design and performs the DMA performance test. It makes use of some classes for operating system abstraction (e.g. **CExAppThread**), also provided by the example application framework code.

- **dma_demo.h**

  This file defines the interface to the code in **dma_demo.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.
3. It launches one thread for each DMA engine that has been selected to participate in the test (as per command-line arguments). For each thread, a context structure, initialized by the main thread, supplies information about DMA transfer size & direction etc., also as per command-line arguments.
4. The main thread then commands all DMA threads to perform DMA transfers continuously for a period specified by command-line arguments, with each thread accumulating a count of bytes transferred, and waits for all DMA threads to finish.
5. The main thread verifies the data transferred by the last DMA transfer of each DMA thread, reporting any verification errors found.
6. The main thread reports DMA transfer throughput statistics, for each DMA engine and in aggregate.
7. Finally, the main thread frees allocated memory, destroys synchronization & thread objects etc. and closes 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, **dma_demo**. The GNU C++ toolchain and associated C and C++ development packages must be installed in the system that is used to build **dma_demo**.

To build **dma_demo**, 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/dma_demo**.

The above procedure builds **dma_demo** 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 **dma_demo** (native 64-bit) and **dma_demo32** (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, **dma_demo.exe**.

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

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

   If using Microsoft Visual Studio 2013, open the solution **(design)/host/proj/win32vs2013/dma_demo.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 **dma_demo.exe**:

1. If using Microsoft Visual Studio 2012, open the solution **(design)/host/proj/win32vs2012/dma_demo.sln**.
   If using Microsoft Visual Studio 2013, open the solution **(design)/host/proj/win32vs2013/dma_demo.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 **dma_demo.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/dma_demo/Debug/**
2012	Debug	x64	**(design)/host/proj/win32vs2012/dma_demo/Debug64/**
2012	Release	Win32	**(design)/host/proj/win32vs2012/dma_demo/Release/**
2012	Release	x64	**(design)/host/proj/win32vs2012/dma_demo/Release64/**
2013	Debug	Win32	**(design)/host/proj/win32vs2013/dma_demo/Debug/**
2013	Debug	x64	**(design)/host/proj/win32vs2013/dma_demo/Debug64/**
2013	Release	Win32	**(design)/host/proj/win32vs2013/dma_demo/Release/**
2013	Release	x64	**(design)/host/proj/win32vs2013/dma_demo/Release64/**

Table 4 : Location of dma_demo.exe

5.3 Building for VxWorks

A **Makefile** is provided for building a downloadable kernel module, **admxrc3DmaDemo.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 **admxrc3DmaDemo.out** for 64-bit SMP Nehalem architecture using the Intel toolchain.

Assuming that the **make** command is successful, the build product is **admxrc3DmaDemo.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 FPGA design

6.1 Using the FPGA design with a Linux host

The demonstration program, **dma_demo**, runs on the host system's CPU and verifies that the FPGA design works as expected. 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 have built the demonstration program as detailed in Section 5.
- 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 PciAddress64Bit=1 PciUseMsi=1
```

The **PciAddress64Bit** and **PciUseMsi** options are performance-enhancing options that respectively enable use of 64-bit PCI Addressing and Message-signalled Interrupts by the driver.

Run the demonstration program

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

```
cd (design)/host/proj/linux
sudo ./dma_demo
```

This should yield output as follows:

```
INFO:  Using 1 DMA engine(s): 0
INFO:  DMA transfer size is 0x80000(524288) byte(s)
INFO:  Testing BlockRAM 0 using Direct Slave channel...
INFO:  No errors were detected in initial test of data transfer to and from BlockR
AMs using Direct Slave channel.
INFO:  Doing DMA performance test...
INFO:  0 data error(s) detected for DMA engine 0
INFO:  DMA engine 0 wrote 8728 MiB to the FPGA in 2.00011 s at 4363.77 MiB/s
INFO:  1 DMA engine(s) transferred 8728 MiB to/from the FPGA at 4363.77 MiB/s
```

The default arguments are to use DMA channel 0 (only), to transfer data from the host to FPGA and to use a DMA transfer size of 0x80000 bytes. These three arguments can be overwritten by specifying them on the command line. For a complete description of the command-line options for the **dma_demo** program, refer to Appendix A. Some examples of running **dma_demo** follow:

- Both DMA engines transferring data from FPGA to host:

  ```
  sudo ./dma_demo 0x3 0x3
  ```

- DMA engine 0 transferring data from FPGA to host, and DMA engine 1 transferring data from host to FPGA:

  ```
  sudo ./dma_demo 0x3 0x1
  ```

- DMA engine 0 transferring data from host to FPGA using a DMA transfer size of 0x12345 bytes:

  ```
  sudo ./dma_demo 0x1 0 0x12345
  ```

6.2 Using the FPGA design with a Windows host

The demonstration program, **dma_demo**, runs on the host system's CPU and verifies that the FPGA design works as expected. 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 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".

(Optional) Apply DMA performance-enhancing tweaks to ADB3 Driver

The ADB3 Driver has some parameters that affect its behaviour, which are located in the Registry under the key

HKLM\System\CurrentControlSet\Services\adb3\Parameters

The DWORD value **PciAddress64Bit** (default 0, as of ADB3 Driver 1.4.16) can be set to 1 in order to make the ADM-XRC-KU1's DMA engines use 64-bit PCI addressing, which can improve DMA performance by reducing or eliminating the need for bounce-buffering in systems with more than 3 GiB of memory.

If the above value is changed, the ADB3 Driver must be restarted in order for the change to take effect. This is most easily accomplished in Windows Device Manager by first disabling and then enabling the ADM-XRC-KU1 device.

NOTE: Installing the ADB3 Driver will overwrite the parameters in the Registry, including **PciAddress64Bit**, with their default values.

Run the demonstration program

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

```
cd (design)\host\bin\win32\x86
dma_demo
```

This should yield output as follows:

```
INFO:  Using 1 DMA engine(s): 0
INFO:  DMA transfer size is 0x80000(524288) byte(s)
INFO:  Testing BlockRAM 0 using Direct Slave channel...
INFO:  No errors were detected in initial test of data transfer to and from BlockR
AMs using Direct Slave channel.
INFO:  Doing DMA performance test...
INFO:  0 data error(s) detected for DMA engine 0
INFO:  DMA engine 0 wrote 8728 MiB to the FPGA in 2.00011 s at 4363.77 MiB/s
INFO:  1 DMA engine(s) transferred 8728 MiB to/from the FPGA at 4363.77 MiB/s
```

The default arguments are to use DMA channel 0 (only), to transfer data from the host to FPGA and to use a DMA transfer size of 0x80000 bytes. These three arguments can be overwritten by specifying them on the command line. For a complete description of the command-line options for the **dma_demo** program, refer to Appendix A. Some examples of running **dma_demo** follow:

- Both DMA engines transferring data from FPGA to host:

  ```
  dma_demo 0x3 0x3
  ```

- DMA engine 0 transferring data from FPGA to host, and DMA engine 1 transferring data from host to FPGA:

  ```
  dma_demo 0x3 0x1
  ```

- DMA engine 0 transferring data from host to FPGA using a DMA transfer size of 0x12345 bytes:

  ```
  dma_demo 0x1 0 0x12345
  ```

6.3 Using the FPGA design in VxWorks

The demonstration program, **admxrc3DmaDemo.out**, runs on the VxWorks target machine 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 1.4.17, as a set of downloadable kernel modules, to the VxWorks target machine, after booting. For this method, please refer to the release notes for **ADB3 Driver 1.4.17 for VxWorks**.
  - (b) By building ADB3 Driver Component 1.4.17 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 1.4.17 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:Y:/example/dma_demo-admxrcku1-trunk/host/proj/vxworks/admxrc3DmaDemo.out
value = -140737478303728 = 0xffff800000996010
```

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 **admxrc3DMADemoIndex** entry point in the DKM, and requires the path to the **(design)/fpga/bit/** directory to be the first argument:

```
-> admxrc3DMADemoIndex "host:Y:/example/dma_demo-admxrcku1-v1_0_0/fpga/bit",(int*)
0,0,1,0,0x80000,2000,10,1
```

.bit file not found error

If you encounter an error when invoking **admxrc3DMADemoIndex** of the form

```
ERROR: Failed to configure target FPGA 0 with 'host:y:/example/dma_demo-admxrcku
1-v1_0_0/fpga/bit/dma_demo-ku115_2i/dma_demo.bit': ADMXRC3_FILE_NOT_FOUND
```

then the likely cause is that there is no a pre-built **.bit** file for the speed grade or temperature grade of the FPGA in the card in use. However, there is no functional difference (for a given FPGA die) between speed grades or temperature grades, so the **.bit** file can be specified directly as the second argument:

```
-> admxrc3DMADemoIndex "","host:Y:/example/dma_demo-admxrcku1-v1_0_0/fpga/bit/dm
a_demo-ku115_2e/dma_demo.bit",0,1,0,0x80000,2000,10,1
```

For details of the available entry points in the **admxrc3DmaDemo.out** DKM and their arguments, refer to Appendix B.

Successfully running the program as described above should yield output of the form:

```
INFO:  Using 1 DMA engine(s): 0
INFO:  DMA transfer size is 0x80000(524288) byte(s)
INFO:  Testing BlockRAM 0 using Direct Slave channel...
INFO:  No errors were detected in initial test of data transfer to and from
       BlockRAMs using Direct Slave channel.
INFO:  Doing DMA performance test...
INFO:  0 data error(s) detected for DMA engine 0
INFO:  DMA engine 0 wrote 1835.5 MiB to the FPGA in 2 s at 917.75 MiB/s
INFO:  1 DMA engine(s) transferred 1835.5 MiB to/from the FPGA at 917.75 MiB/s
```

Appendix A: Running the demonstration program in Linux & Windows

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

```
dma_demo [option ...] [DMA channels] [DMA directions] [DMA transfer size]
```

where the the positional arguments are as follows (in this order, unless omitted):

- **[DMA channels]**

  This positional argument is a bitmask that specifies which DMA engines participate in the test. Bit 0 corresponds to DMA engine 0, and bit *N* corresponds to DMA engine *N*. If a particular bit is 1, the corresponding DMA engine is included.

  If omitted, the default value is 1, which corresponds to DMA engine 0 (only).

  Examples:

  - **1**

    DMA engine 0 (only) participates.

  - **3**

    DMA engines 0 & 1 participate.

- **[DMA directions]**

  This positional argument is a bitmask that specifies the direction of data transfer for each participating DMA engine. Bit 0 corresponds to DMA engine 0, and in general bit *N* corresponds to DMA engine *N*. If a particular bit is 1, the corresponding DMA engine transfers data from the FPGA to the host; otherwise from the host to the FPGA. Note that if the corresponding bit of **[DMA channels]** is 0, a given bit of **[DMA directions]** is ignored.

  If omitted, the default value is 0, which corresponds to all participating DMA engines transferring data from the host to the FPGA.

  Examples:

  - **1**

    DMA engine 0 transfers data from FPGA to host; the other(s) from host to FPGA.

  - **3**

    DMA engines 0 & 1 transfer data from FPGA to host.

- **[DMA transfer size]**

  This positional argument is the DMA transfer size used for all participating DMA engines. It must be in the inclusive range 1 to (size of per-DMA-engine BlockRAM), i.e. in the range [1, 0x80000].

  If omitted, the default value is (size of per-DMA-engine BlockRAM), i.e. 0x80000 (512 kiB).

  Examples:

  - **1**

    DMA transfers are a single byte each.

  - **12345**

    DMA transfers are 12345 bytes each.

  - **0x4000**

    DMA transfers are 16384 bytes / 16 kiB each.

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

- **-duration** ***<duration of performance test, in milliseconds>***

  This option specifies the duration of the performance test, in milliseconds.

  If omitted, the value is 2000, which is chosen as reasonable compromise between time taken and minimising variance from one run to another.

  Examples:

  - **-duration 1500**

    The performance test lasts for 1500 milliseconds.

  - **-duration 0xEA60**

    The performance test lasts for 0xEA60 milliseconds, i.e. one minute.

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

  This option displays a brief help message.

- **-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.

- **-maxerr** ***<maximum number of errors to display>***

  This option specifies the maximum number of data verification errors to be displayed in detail, in phases of the test where data that has been transferred is verified for correctness. If more than the specified number of errors occur, a message is displayed to indicate that further errors have occurred, but their details are suppressed.

  If omitted, the value is 10.

  Examples:

  - **-maxerr 5**

    Display up to 5 verification errors in detail.

  - **-maxerr 0x20**

    Display up to 0x20 (32) verification errors in detail.

- **-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).

- **-verify *<bool>***

  This option enables or disables data verification, and should normally be left at the default value of **true**.

  If omitted, the value is **true**.

  Examples:

  - **-verify false**, **-verify 0**

    Disable data verification.

  - **-verify true**, **-verify 1**

    Enable data verification.

Appendix B: Demonstration program entry points in VxWorks

The demonstration program can be invoked via two entry points in the **admxrc3DmaDemo.out** DKM. These entry points are defined by the header file, **(design)/host/src/dma_demo.h**, as follows:

```
int
admxrc3DMADemoIndex(
  const TCHAR* pBitPath,
  const TCHAR* pBitFile,
  unsigned int index,
  uint32_t     dmaEngineMask,
  uint32_t     dmaDirectionMask,
  uint32_t     dmaTransferSize,
  unsigned     int durationMs,
  unsigned     int maxErrorDisplayed,
  int          bVerifyData);

int
admxrc3DMADemoSN(
  const TCHAR* pBitPath,
  const TCHAR* pBitFile,
  uint32_t     serialNumber,
  uint32_t     dmaEngineMask,
  uint32_t     dmaDirectionMask,
  uint32_t     dmaTransferSize,
  unsigned     int durationMs,
  unsigned     int maxErrorDisplayed,
  int          bVerifyData);
```

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**.

- **admxrc3DMADemoIndex**

  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.

- **admxrc3DMADemoSN**

  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 + "/dma_demo-**<device>_<speed><tempgrade>**[**_<step>**]/dma_demo.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 KU115-2E device, the full path of the **.bit** file is constructed as:

  pBitPath + "/dma_demo-ku115_2e/dma_demo.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 **admxrc3DMADemoIndex** 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 **admxrc3DMADemoSN** entry point, this parameter specifies the serial number of the reconfigurable computing card to use.

- **dmaEngineMask**

  This is a bitmask that specifies which DMA engines participate in the test. Bit 0 corresponds to DMA engine 0, and bit *N* corresponds to DMA engine *N*. If a particular bit is 1, the corresponding DMA engine is included.

  Examples:

  - **1**

    DMA engine 0 (only) participates.

  - **3**

    DMA engines 0 & 1 participate.

- **dmaDirectionMask**

  This is a bitmask that specifies the direction of data transfer for each participating DMA engine. Bit 0 corresponds to DMA engine 0, and in general bit *N* corresponds to DMA engine *N*. If a particular bit is 1, the corresponding DMA engine transfers data from the FPGA to the host; otherwise from the host to the FPGA. Note that if the corresponding bit of **dmaEngineMask** is 0, a given bit of **dmaDirectionMask** is ignored.

  Examples:

  - **1**

    DMA engine 0 transfers data from FPGA to host; the other(s) from host to FPGA.

  - **3**

    DMA engines 0 & 1 transfer data from FPGA to host.

- **dmaTransferSize**

  This argument is the DMA transfer size used for all participating DMA engines. It must be in the inclusive range 1 to (size of per-DMA-engine BlockRAM), i.e. in the range [1, 0x80000].

- **durationMs**

  This parameter specifies the duration of the performance test, in milliseconds.

  A suggested value is 2000.

- **maxErrorDisplayed**

  This parameter specifies the maximum number of data verification errors to be displayed in detail, in phases of the test where data that has been transferred is verified for correctness. If more than the specified number of errors occur, a message is displayed to indicate that further errors have occurred, but their details are suppressed.

  A suggested value is 10.

- **bVerifyData**

This parameter enables (if nonzero) or disables (if zero) data verification, and should normally be given a nonzero value.

If **pBitFile** is not NULL, it overrides any value passed for **pBitPath**. Table 5 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 5 : Interaction of pBitPath and pBitFile

Appendix C: Makefile variables in VxWorks

The **Makefile** for building the downloadable kernel module **admxrc3DmaDemo.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 **admxrc3DmaDemo.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 Jun 2016	1.0	Initial version.