Exercising Vitis AI Applications on Alpha Data Boards

A. Adetomi

Introduction

This white paper provides an educational overview of the requirements for exercising Vitis AI applications on Alpha Data boards. It documents the Vitis platform build process and the flow for a typical Vitis AI application build and execution.

The Xilinx Deep Learning Processor Unit (DPU) is a configurable computation engine intended for convolutional neural networks. The degree of parallelism utilized in the engine is a design parameter that is dependent on the application. It includes a set of highly optimised instructions, and supports most convolutional neural networks, such as VGG, ResNet, GoogleNet, YOLO, SSD, MobileNet, and FPN. Xilinx Vitis AI is a development stack for AI inference on Xilinx hardware platforms.

Vitis AI applications are built on top of extensible platforms by integrating one or more DPUs as kernel. These platforms integrate hardware for supporting acceleration kernels in the device, and software for a target running Linux and the Xilinx Runtime (XRT) library.

Vitis Base Platform

In order to exercise Vitis capabilities and topologies on a target Alpha Data board, a Vitis base platform is required. One seeks a base platform that will be capable of running Vitis acceleration applications (e.g., Vector Addition) and Vitis AI applications (e.g., Resnet50) in addition to general embedded software applications. Essentially, a base platform is made extensible by providing multiple clock and memory interfaces such that when integrating an acceleration block or a DPU, there is flexibility on clock frequency and memory bandwidth depending on the application requirements. In addition, it should include sufficient software components such that the Vivado-generated implementation result (BIT file) and the PetaLinux-generated images for the platform should be able to successfully boot to the Linux console.

Figure 1 is an example base platform, showing the DDR interfaces already provisioned from the processing system but not connected to any kernel. Note that there are other components (e.g., clocking wizard for generating the clocks) not shown in this interface view.

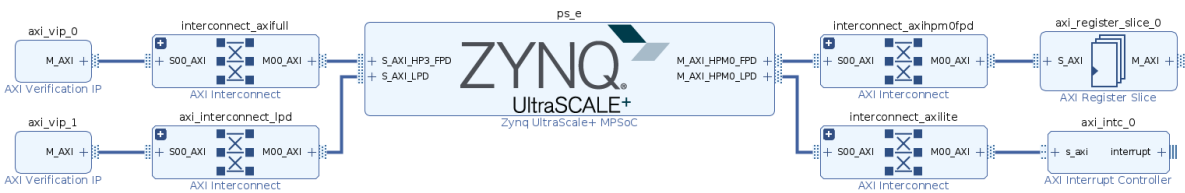

Figure 1 : Example Base Platform (Interface View)

Hardware Configurations

The platform for running Vitis AI applications needs to provide all the clocks and the interfaces needed for both kernel control and memory access. For instance, a Vitis AI application would integrate DPU as RTL kernel; and

require one interrupt, two clocks, and as many AXI HP interfaces as can be supported by the target device, noting that the DPU is resource and memory-intensive.

Software Configurations

To support the Vitis application acceleration development flow, embedded platforms must run Linux, with XRT integrated into the rootfs. Vitis AI software framework can also control the DPU with XRT. The XRT facilitates communication between the host application code (running on the Arm processor) and the accelerated kernels deployed on the reconfigurable fabric of the device.

The XRT comprises userspace and kernel driver components. ZOCL is the kernel module (for MPSoC devices) that communicates with acceleration kernels and is responsible for memory management, execution control, DMA operations, device management/monitoring, and complete image download. The ZOCL requires a node in the device tree and this is added in the system-user.dtsi file. In the userspace, host applications can use the OpenCL API, to control acceleration kernels.

Therefore, the platform needs to provide the XRT, ZOCL packages and other Vitis AI dependencies. The following also have to be added: GCC compilers, for application native compilation; and mesa-mega driver, for Vitis AI demo applications.

Targeting Alpha Data Boards

A base platform that targets the Alpha Data board in question is required. Base platforms and Vitis AI reference designs for Alpha Data boards are available on the Alpha Data ShareFile site - contact Alpha Data for details. The platforms for our MPSoC boards come with the hardware and software configurations in Table 1 and Table 2 respectively.

Configuration	Values	Details
Clocks for Kernels	150MHz, 300MHz, 75MHz, 100MHz, 200MHz, 400MHz, 600MHz	Clocks are synchronous to each other
PS DDR Interfaces for Kernels	HP0, HP1, HP2, HP3, HPC0, HPC1, LPD	All the ports share the same PS DDR
Interrupts	32 Interrupts are enabled from PL Kernel to PS	

Table 1 : Hardware Configurations

Configuration	Values
Additional Kernel Configurations [user.cfg]	CONFIG_CONSOLE_LOGLEVEL_DEFAULT=1

Table 2 : Software Configurations (continued on next page)

Configuration	Values
Additional RootFS Components [rootfs_config]	DNF e2fsprogs-resize2fs parted libmali-xlnx: disabled xrt, xrt-dev and zocl opencl-clhpp opencl-headers libdrm, libdrm-tests and libdrm-kms packagegroup-petalinux-audio packagegroup-petalinux-gstreamer packagegroup-petalinux-matchbox packagegroup-petalinux-opencv packagegroup-petalinux-v4lutils packagegroup-petalinux-vitisai packagegroup-petalinux-x11 imagefeature-package-management auto-login
Device Tree Modifications [system-user.dtsi]	Add zocl node for XRT Disable default dtg generated axi intc PL node and add the custom node instead
Interrupts [system-user.dtsi]	32 Interrupts are enabled from PL Kernel to PS

Table 2 : Software Configurations

Building a Vitis Base Platform

A typical Vitis platform creation flow involves the creation of Vivado hardware design, generation of the XSA, creation of the software components with PetaLinux, packaging of the platform, and testing.

Building the base platform requires the Vitis software, which includes the Vivado Design Suite for building the hardware design. See Xilinx Vitis Embedded Installation-Requirements (UG1400) for more details about supported operating systems. Building the Linux image requires PetaLinux and a compatible Linux operating system.

The choice of the hardware architecture of the base platform is user-dependent but it is required to have the interfaces needed for clock, kernel control, and memory access. Nevertheless, a base platform should be made as generic as possible within the context of the different Vitis acceleration and AI applications for which it is intended.

Platform Naming

Vitis platform naming should follow the Xilinx's Platform Naming Convention as follows:

```
<Vendor>_<Board>_<Feature>_<Supported Vitis Tool Version>_<Release Version>
```

Where:

- **<Vendor>** is the board vendor. For all pre-built platforms created by Alpha Data, the string "ad" is used.
- **<Feature>** is the special function of the platform. For instance, the value "base" indicates that all required resource for an acceleration application have been included, whereas the value "dfx" indicates support for Xilinx Dynamic Function eXchange (DFX).
- **<Supported Vitis Tool Version>** is the specific version of the targeted Vitis development platform and also indicates the version of the Vivado Design Suite tools used to create the pre-built platform.

- **<Release Version>** is the release version of the platform, with the first version being 1.

Based on this naming convention, the following are platform name examples from Alpha Data: *ad_9z2_base_202020_1* and *ad_9z5_base_202020_1*.

Download the Source and Retarget the Board Properties and PS Configuration

Go to https://github.com/Xilinx/Vitis_Embedded_Platform_Source and clone the official platform build source (e.g., xilinx_zcu102_base). We have tested with the 2020.2 branch.

Change part selection in "hw/xsa_scripts/xsa.tcl", and where board files are available for the target board, add these as well by setting the "board.repoPaths" parameter.

In addition, change the processing system (ps_e) properties in the "hw/xsa_scripts/dr.bd.tcl". One way to get going quickly is to start from a platform project for the ZCU102 board so that the block design is successfully generated. Then change the device selection and also re-customise the processing system. After this, the PS settings can be exported for use in the "hw/xsa_scripts/dr.bd.tcl". During the PS configuration, ensure that USB, Ethernet, DDR, and other I/O and memory settings are updated for the target board.

PetaLinux configurations may also be changed in the folder "<project_root>/sw/petalinux". For instance, you may find that there is not enough space on the target device to run the application. An extra rootfs space of 1GB is enough, and can be added in the PetaLinux configuration (<project_root>/sw/petalinux/project-spec/meta-user/conf/petalinuxbsp.conf) by adding the following:

```
IMAGE_ROOTFS_EXTRA_SPACE = "1048576"
```

Platform Build Flow

The platform build process is entirely scripted and is only supported in Linux environments as it involves cross-compiling Linux. However, it is possible to build inside a VM or Docker container.

Also note that the default PetaLinux configuration expects the TMPDIR to be local to the system. This will *not* work if building on a Network File System (NFS) as Yocto will error out, in which case PetaLinux should be updated to change the build area to a locally mounted hard drive. However, note that the same location should not be configured as TMPDIR for two different PetaLinux projects as it can cause build errors.

The default TMPDIR is set with the property CONFIG_TMP_DIR_LOCATION and can be changed in the config file "<project_root/sw/petalinux/project-spec/configs/config".

To build the platform, extract the sources from the downloaded folder and take the following steps:

1: Set up the Vitis environment:

```
source <Vitis_install_dir>/settings64.sh
```

2: Source the PetaLinux configuration script and go to the project directory:

```
source <PetaLinux_install_dir>/settings.sh
```

```
cd <project_root>
```

3: Run 'make' to generate the platform. The following command will build all the hardware and software components:

```
make all COMMON_RFS_KRNL_SYSROOT=FALSE
```

Check the Makefile in the root folder for other build flags. By default, the unmodified Makefile will install the platform to "platform_repo/<platform name>/export/<platform name>/".

Note
To remove the generated files, run the command 'make clean'. This may be required if the build has to be restarted from a clean state after a build error. For instance, a "$RDI_PROG" "$@" synthesis crash may happen if Vivado runs out of memory. This may be a result of having too high a value for the number of jobs, depending on your system setup. In this case, consider reducing the number of jobs in "<project_root>/hw/xsa_scripts/pfm_decls.tcl". Look for the line containing "launch_runs", reduce the default value of 8, clean the generated files, and rerun step 3.

Build the PetaLinux SDK

Embedded platforms require a sysroot to cross-compile the host application for the Vitis application acceleration flow. Running sdk.sh extracts and installs the sysroot. The option -d allows the choice of where to install the sysroot. This package also provides pre-compiled kernel image and rootfs (see the image folder in the prebuilt directory).

The sysroot can be added to the Makefile or specified in the 'make' command. For example, in the Makefile point <SYSROOT> to "<SDK_install_dir>/sysroots/aarch64-xilinx-linux", which is generated when running sdk.sh.

It is possible to use the common Linux components (prebuilt linux kernel, boot files, root filesystem and sdk.sh script to generate sysroot) provided by Xilinx, but they can be built from scratch using PetaLinux.

To build and install the SDK, take the following steps:

1: Source the PetaLinux configuration script, if required:

```
source <PetaLinux_install_dir>/settings.sh
```

2: Go to the project root, if not already there:

```
cd <project_root>
```

3: Build and install the SDK to "<project_root>/platform_repo/sysroot/".

```
make petalinux_sysroot
```

Test the Platform

Platforminfo Test

The platform should have a proper platforminfo report for clock and memory information. The following command can be run on the development machine to get a console output similar to that in Figure 2:

```
platforminfo <path/to/platform>.xpfm
```

XRT Basic Test

The xbutil (Xilinx Board Utility) *query* command should be able to run on the target board and properly report platform information. The xbutil is a standalone command line utility that is included with the XRT installation package. It includes multiple commands to validate and identify the installed device along with additional details including DDR, shell name (DSA), and system information.

```
xbutil query
```

Vector Addition (Vadd) Test

Vector Addition is a simple acceleration PL kernel, which can be used for a functional test of the generated platform. It requires one clock, one interrupt, one M_AXI for kernel control and one S_AXI for memory access. Running the Vadd application can check the AXI control bus, memory interface and interrupt setting in platform are working properly. A valid Vadd sample application and xclbin should print "TEST PASSED" on the console when run on the target. Build instructions for the Vadd application can be found on GitHub (Vitis Acceleration

Example - Hello World).

```
====================================
Hardware Platform (Shell) Information
====================================
Vendor:                           alpha-data.com
Board:                            ad_9z5_base_202020_1
Name:                             ad_9z5_base_202020_1
Version:                          202020.1
Generated Version:                2020.2_AR72992
Hardware:                         1
Software Emulation:               1
Hardware Emulation:               1
Hardware Emulation Platform:      0
FPGA Family:                      zynquplus
FPGA Device:                      xqzu19eg
Board Vendor:
Board Name:
Board Part:

=================
Clock Information
=================
  Default Clock Index: 0
  Clock Index:         0
    Frequency:         150.000000
  Clock Index:         1
    Frequency:         300.000000
  Clock Index:         2
    Frequency:         75.000000
  Clock Index:         3
    Frequency:         100.000000
  Clock Index:         4
    Frequency:         200.000000
  Clock Index:         5
    Frequency:         400.000000
  Clock Index:         6
    Frequency:         600.000000

=================
Memory Information
=================
  Bus SP Tag: HP0
  Bus SP Tag: HP1
  Bus SP Tag: HP2
  Bus SP Tag: HP3
  Bus SP Tag: HPC0
  Bus SP Tag: HPC1
  Bus SP Tag: LPD
```

Figure 2 : Example Console Output for Platforminfo (Part View)

Vitis AI Application

This section provides instructions for targeting the Xilinx Vitis AI 1.3.2 flow to Alpha Data's Vitis platforms. The instructions and scripts provided are adapted from the Xilinx® DPU targeted reference design (TRD) , which provides instructions on how to use the DPU with a Xilinx SoC platform to build and run deep neural network applications.

> **Note**
> The build process provided here targets the Vitis AI 1.3.2 and the corresponding Vitis 2020.2 version. There may be some variations in the build process when targeting other versions.

The compilation of the application will require XRT to be installed on the development machine. Install the XRT 2020.2 (https://github.com/Xilinx/XRT/tree/2020.2), if not already installed. The XRT installation steps can be found here: https://xilinx.github.io/XRT/2020.2/html/install.html .

Download the Project Source

Download the Vitis AI source from https://github.com/Xilinx/Vitis-AI/tree/1.3.2/dsa , noting the selected branch while doing so. Figure 3 shows the structure of the DPU-TRD folder. Source files are provided for building the platform from scratch. The TRD supports both the Vitis and Vivado flows. However, this white paper covers only the Vitis flow.

Figure 3 : Project Folder Structure

Configuring the DPU

The DPU IP provides some user-configurable parameters to optimise resource utilisation and customise different features. Different configurations can be selected for DSP slices, LUT, Block RAM (BRAM), and UltraRAM utilisation based on the amount of available programmable logic resources. There are also options for addition functions, such as channel augmentation, average pooling, depthwise convolution.

For more details about the DPU, please refer to DPUCZDX8G for Zynq UltraScale+ MPSoCs, PG338 (v3.3)

Vitis Configuration File

The Vitis tool uses a configuration file (instead of the previous command line switches) to control the compiler and linker behaviour, where multiple options or properties can be grouped into sections. The following are relevant section names and their usage:

- **[clock]**: To specify the clock option using the clock ID or the clock frequency
- **[connectivity]**: To specify system topology such as the number of kernels and port connections
- **[vivado]**: To control Vivado properties and parameters
- **[advanced]**: To gain fine-grain control over the hardware generated

Here is the file path of the configuration file: "<project_root>/prj/Vitis/config_file/prj_config"

Set the DPU Core Number

The DPU core number defines the number of DPUs that will be integrated. For example, adding "nk= DPUCZDX8G:2" under the [connectivity] section in the Vitis configuration file will set the number of DPUs to 2. If this property is deleted, the project will integrate one DPU. The core number can be changed to make use of as many DPUs as needed. However, it should be noted that the DPU is resource-intensive, consuming lots of LUTs and RAMs. By implication, using 3 or more DPUs may cause hardware timing closure issues.

Modify the DPU Parameters

The file <project_root>/prj/Vitis/dpu_conf.vh can be modified in order to change the configuration of the DPU. Table 3 lists the parameters that can be modified, the available options, and the defaults for the TRD.

Parameter	Options	Default
Architecture	B512, B800, B1024, B1152, B1600, B2304, B3136, B4096	B4096
URAM Number	URAM_ENABLE, URAM_DISABLE	URAM_ENABLE
RAM Usage	RAM_USAGE_HIGH, RAM_USAGE_LOW	RAM_USAGE_LOW
Channel Augmentation	CHANNEL_AUGMENTATION_ENABLE, CHANNEL_AUGMENTATION_DISABLE	CHANNEL_AUGMENTATION_ENABLE
DepthwiseConv	DWCV_ENABLE, DWCV_DISABLE	DWCV_ENABLE
AveragePool	POOL_AVG_ENABLE, POOL_AVG_DISABLE	POOL_AVG_ENABLE
ELEW MULT	ELEW_MULT_ENABLE, ELEW_MULT_DISABLE	ELEW_MULT_DISABLE
ReLU Type	RELU_RELU6, RELU_LEAKYRELU_RELU6	RELU_LEAKYRELU_RELU6
DSP Usage	DSP48_USAGE_HIGH, DSP48_USAGE_LOW	DSP48_USAGE_HIGH
Low Power Mode	LOWPOWER_ENABLE, LOWPOWER_DISABLE	LOWPOWER_DISABLE
Device	MPSOC, ZYNQ7000	MPSOC

Table 3 : DPU Parameters, Options, and Defaults

For the URAM numbers, Xilinx has recommendations for different DPU architectures as indicated in Table 4. The URAM numbers can also be adjusted according to the resource usage of the entire project. The default setting in dpu_conf.vh is for B4096. To change the URAM numbers, locate def_UBANK_IMG_N, def_UBANK_WGT_N, and def_UBANK_BIAS in the dpu_conf.vh file and modify as needed.

	B512	B800	B1024	B1152	B1600	B2304	B3136	B4096
U_BANK_IMG	2	2	4	2	4	4	4	5
U_BANK_WGT	9	11	9	13	11	13	15	17
U_BANK_BIAS	1	1	1	1	1	1	1	1

Table 4 : URAM Number Recommendations

The TRD supports the softmax function. With reference to the make command in step 3 of section Building the Hardware Design, to use only the DPU, run "make KERNEL=DPU". Otherwise, to use the DPU and Softmax, use "make KERNEL=DPU_SM".

Specify Clocks

The DPU requires two clocks: clk and clk2x, e.g., a combination of 150 MHz and 300 MHz. The clocks are specified under the [clock] section in the Vitis configuration file.

> **Note**
> The information about the clocks in the platform can be retrieved by running the platforminfo command on the development machine. See Figure 2 for an example.

Specify Connectivity for DPU Ports

The DPU ports are specified under the [connectivity] section in the Vitis configuration file.

> **Note**
> The information about the ports in the platform can be retrieved by running the platforminfo command on the development machine. See Figure 2 for an example.

Fixing Timing Issues

If the project has timing issues, the Vivado implementation strategy can be changed under the [vivado] section in the Vitis configuration file. The possible strategies can be found in Vivado Implementation Strategies . Another option is to reduce the clock frequency or the number of integrated DPUs if there is no strategy that ensures timing closure.

Building the Hardware Design

To build the hardware design for the Vitis AI application, that is, integrate the DPU into the Vitis platform, take the following steps:

1: Source the Vitis and XRT setup scripts:

```
source <Vitis_install_dir>/Vitis/<version>/settings64.sh
source <XRT_install_dir>/xilinx/xrt/setup.sh
```

2: Go to the Vitis project directory:

```
cd <project_root>/prj/Vitis
```

3: Build the hardware:

```
export EDGE_COMMON_SW=<path/to/rootfs/and/kernel/image>
```

```
export SDX_PLATFORM=<path/to/base/platform>.xpfm

make KERNEL=DPU_SM DEVICE=<device name>
```

A pre-generated matching model file for the default DPU settings is in the <project_root>/app/ path. If the DPU settings are changed, the model needs to be recompiled.

SD card files are generated in <project_root>/prj/Vitis/binary_container_1/sd_card.

Figure 4 is an example hardware generated for Vitis AI after following the above steps, with two DPUs integrated.

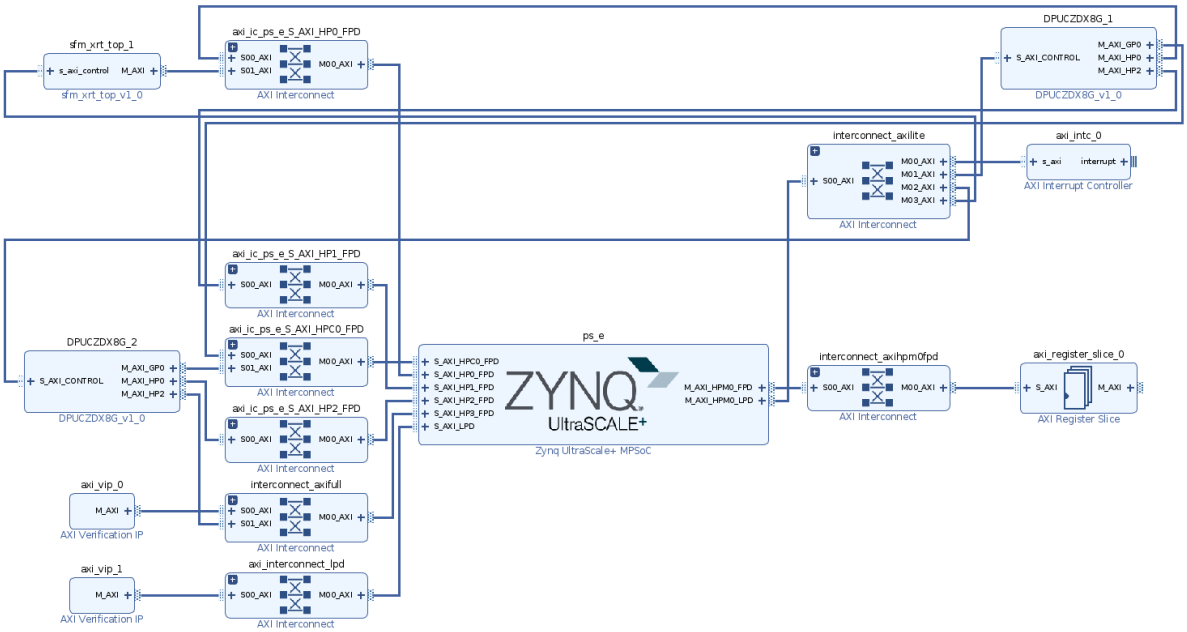

Figure 4 : Example Vitis AI Hardware (Interface View)

Running the Resnet50 Example

After the build flow, all the related files are packaged in "<project_root>/prj/Vitis/binary_container_1/sd_card.img" by the Vitis tools. The user can use the balenaEtcher tool , or the dd utility on Linux, to write this image onto an SD card.

1: Get the img folder from https://github.com/Xilinx/Vitis-AI/tree/1.1/DPU-TRD/app and copy it to "<project_root>/app".

2: Copy the directory "<project_root>/app" to SD card.

3: Boot the board.

4: After the Linux boot, run:

```
cp -r /mnt/sd-mmcblk0p1/app/samples ~

cp /mnt/sd-mmcblk0p1/app/model/resnet50.xmodel ~

cp -r /mnt/sd-mmcblk0p1/app/img ~

env LD_LIBRARY_PATH=samples/lib \

XLNX_VART_FIRMWARE=/media/sd-mmcblk0p1/dpu.xclbin \

samples/bin/resnet50 img/bellpeppe-994958.jfif
```

An example console output is shown in Figure 5, which is for the ADM-VPX3-9Z5 board.

```
root@ad-9z5-2020_2:~# cp -r /mnt/sd-mmcblk0p1/app/samples ~
root@ad-9z5-2020_2:~# cp /mnt/sd-mmcblk0p1/app/models/resnet50.xmodel ~
root@ad-9z5-2020_2:~# cp -r /mnt/sd-mmcblk0p1/app/img ~
root@ad-9z5-2020_2:~# env LD_LIBRARY_PATH=samples/lib \
> XLNX_VART_FIRMWARE=/media/sd-mmcblk0p1/dpu.xclbin \
> samples/bin/resnet50 img/bellpeppe-994958.jfif
score[945]  =  0.992235     text: bell pepper,
score[941]  =  0.00315807   text: acorn squash,
score[943]  =  0.00191546   text: cucumber, cuke,
score[939]  =  0.000904801  text: zucchini, courgette,
score[949]  =  0.00054879   text: strawberry,
root@ad-9z5-2020_2:~#
```

Figure 5 : Example Console Output for Resnet50

References

The following are useful references:

- Getting Started with Vitis
- Vitis Compiler Command and Xilinx Utilities
- Kernel Interface Requirements
- Vitis AI Overview
- Xilinx® DPU targeted reference design (TRD)

Revision History

Date	Revision	Nature of Change
24/05/2022	1.0	Initial release

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