Space Tolerant CNN FPGA Deployment
Part 4: Deployment on the ADA-SDEV-KIT3

A. C. McCormick

Space Tolerant CNN FPGA Deployment, Part 4

This paper is the final part of a four-part series of white papers providing an educational overview of the issues surrounding the deployment of Convolutional Neural Network solutions on FPGAs in a radiation susceptible environment. The first part documented a practical CNN processing core, which can be used to implement a wide range of CNN solutions. The second part, discussed the Space Hardening of that core, adding in Triple-Mode Redundancy for radiation effect tolerance to control path circuitry. This third part documented the higher level control structures necessary to move data to and from the CNN core and dynamically reconfigure its operation to match the functional requirements of a parts of the model. This final fourth part of this series documents the deployment of this FPGA solution on the Alpha Data ADA-SDEV-KIT3 Space Development kit for the Xilinx XQRKU060 FPGA device.

ADA-SDEV-KIT3 Development Board

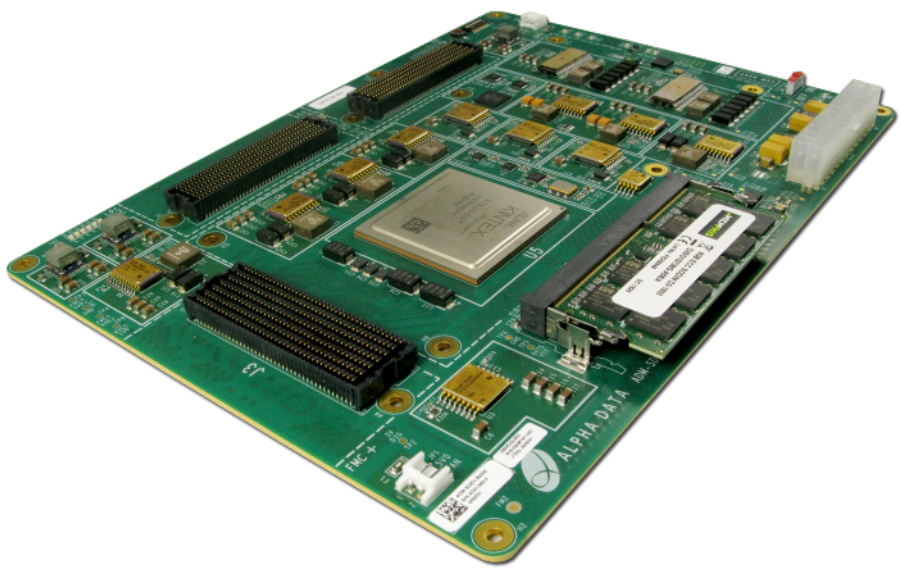
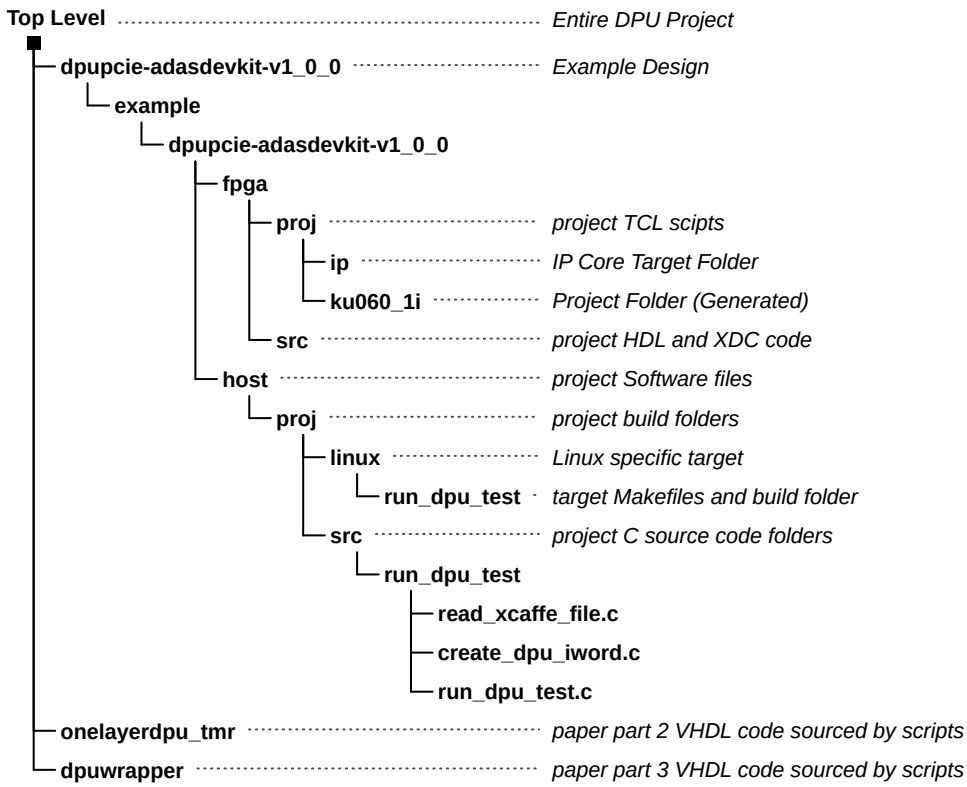


Figure 1 : ADA-SDEV-KIT3

The Alpha Data ADA-SDEV-KIT3 is a development board for customers wishing to deploy applications on the XQRKU060 Space Grade FPGA. The board was developed as a reference design by Alpha Data in collaboration with Xilinx and Texas Instruments. The board features parts with pin compatible Space Grade alternatives available, providing a reference for Space at laboratory grade component cost. This includes full EM compatible (non-screened, only functionally tested at lab temperature) Space Grade power supply chips from TI and the almost pin compatible Xilinx XCKU060 FPGA device. This board also provides a selection of FMC

options to allow customers to configure their IO requirements, including the Config-FMC, which includes a PCIe compatible IPASS connector which will be used to transfer data to and from the DPU design. The board also features DDR3 memory on a SODIMM, which will also be utilized by the DPU design.

The example code structure for this part of the paper is based on the reference designs for the ADA-SDEV-KIT3. Therefore it has a more complex directory structure when compared with the other parts of this paper, with directories for FPGA designs, source code and projects. There are also directories for host software since a PCIe connected host will be used for overall system control and data transfer and evaluation. The directory structure is as follows:



The code developed for the earlier parts of the paper must be included in the indicated folders to be sourced by the IP packaging scripts.

DPU IP Packaging

In order to combine the DPU design developed in the first 3 parts of this paper with other IP necessary to build a board level system, the packaging up of the DPU IP is required. By encapsulating the VHDL design as an XACT IP core it can be used along side other off-the-shelf IP in the Xilinx IP Integrator, block diagram capture tool that sits within the Vivado design suite.

The design, as left at the end of the simuations in part 3, still requires a few minor modifications to make it suitable for integration with other IP cores. The control and status monitoring signals need to be wrapped up into an AXI4 compatible form, and made accessible as registers. This will allow a remote processor to configure and trigger the DPU runs and monitor the performance. The IPI tool and the other IP required to interface to the outside world does not support TMR and is not aware of the TMR types used and therefore the TMR DPU core needs to be wrapped with signals resolved to standard logic vector types.

Therefore in the source code 2 new files are added: *reg_bank_axi4l.vhd* which provides a generic AXI4 Lite

register bank, with programmable bit fields that can be connected to the DPU core control and status signals and *dpu_top.vhd* which connects and wraps up the DPU core and register bank and resolves any TMR signals.

The IP is packaged using the script *mkip.tcl* which pulls in the existing files from the folders from previous parts of the paper, and combines them with the 2 new files. The DataMover IP cores are also generated and included in the project. This is then all packaged up and saved as an IP core *dpu_top_v1_0.zip* in the *ip* subfolder for inclusion in the main project.

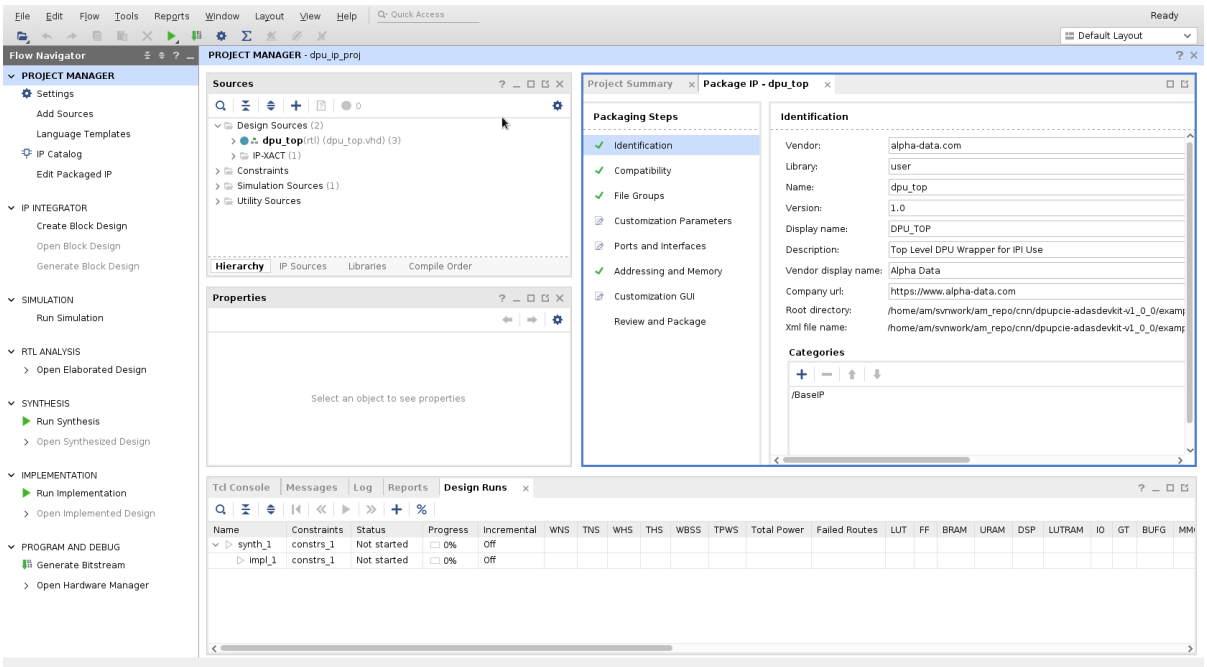


Figure 2 : Packaging DPU IP Using Vivado

IP Integration

Since we now have the DPU packaged up neatly as an IP Core it can now be used with the Xilinx IP integrator tool within Vivado. The block diagrams can be schematically edited, connecting up signals via the GUI. The block diagrams can also be built using TCL scripts for a more traceable and repeatable design flow, that can work easily with version control systems. The project folder contains scripts for generating a project to build the FPGA bistream (*mkxpr-ku060_1i.tcl*), and this calls the script that generates the block diagram *mkbd.tcl*

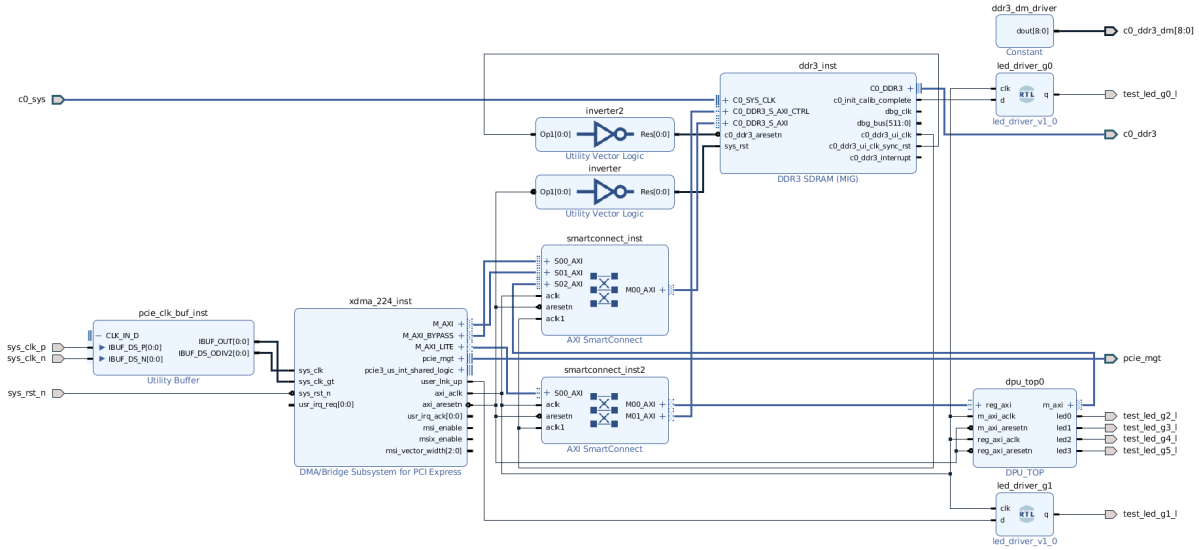


Figure 3 : Vivado IPI Integrator Block Diagram

The scripts are based on the PCIe reference design for the ADA-SDEV-KIT and use a PCIe configuration based on that example. They also instantiate the DDR3 core, and set it up with the same parameters as used in the DDR3 example from the ADA-SDEV-KIT reference designs. The script then adds in the DPU core and connects up the AXI4 buses so that there is high performance access from both the host (PCIe core) and the DPU cores memory port to the DDR3 interface IP block. AXI4 bus connection is also set up to enable the host (via the PCIe core) to access the AXI-Lite register interface on the DPU. Additionally the block diagram connects up some of the LEDs on the board to show the status of the application. One indicates that the PCIe is online and another that the DDR3 memory has trained correctly. The other 4 LEDs connect to status bits that show the state of the DPU.

State	LED2	LED3	LED4	LED5
Idle	On	On	On	On
Read Instruction Word Cmd	On	On	On	Off
Wait for Instruction Word	Off	On	On	On
Reading Instruction Word	On	On	Off	On
Read Weights Cmd	On	On	Off	Off
Wait for Weights	Off	On	On	Off
Reading Weights	On	Off	On	On
Start Writer Cmd	On	Off	On	Off
Start Writer 2 Cmd	Off	On	Off	Off
Start Reader 2 Cmd	Off	On	Off	On
Start DPU Cmd	On	Off	Off	On
DPU Active	On	Off	Off	Off
DPU Pausing	Off	On	Off	Off
Error State	Off	Off	Off	Off

Table 1 : Data Files

Host Software

While the DPU runs autonomously on data and a program placed in SDRAM, some higher level control is still required to set up the contents of SDRAM, generate the DPU program, transport the input and output data and start and monitor the DPU until it completes.

The example host software is however a simple example that aims to replicate the behaviour of the test bench tests used in previous papers, reading in the text only Caffe file to get the weights and bias definitions for the YoloV3 network and use them to configure DPU to run the various different layers.

The example code is split into 3 files: *read_xcaffe_file.c, create_dpu_iword.c* and the top level *run_dpu_test.c*

read_xcaffe_file.c is a port from VHDL to C of the test bench function with the same name. This function extracts from a text only Caffe file, the weight and bias data required for a specified network layer. This is then stored in the required 16 bit integer format in host memory ready to transfer to the FPGA attached memory. This function does not parse the Caffe file for the network structure. This is hard-coded into the top level code.

create_dpu_iword.c exports a function of the same name that assembles the 128 byte instruction word for the DPU based on the parameters passed in. These parameters come from the network description hard-coded into the top layer.

run_dpu_test.c is the top level main function that tests the DPU network on the FPGA. This design hard codes in the YoloV3 network structure in the order to used by the DPU. While based on the Caffe model, this information may not be automatically be extractable, and there are cases where one layer in the Caffe model maps to multiple sequential runs of a subset of neurons on that layer. This is currently mapped by hand. The resulting DPU program runs as 29 sequential layers - where the Caffe description only features 21 (some of these 21 are MaxPool layers that are absorbed into the preceding convolutional layer in the DPU). As well as the structure being hard coded here, the memory map for layer inputs and outputs is also mapped manually. In cases where the Caffe Model Layer is split into multiple sequential runs of 128 neurons, the memory writes are interleaved.

The test program has some command line parameters to simplify any required debug. These allow the number of DPU layers actually run to be controlled, which can be useful for spotting which layer causes a lock up, due to incorrect specification of the instruction word. A second option allows the layers to be run sequentially by the CPU, rather than as a linked list, where the CPU waits for the last layer to complete. This can also isolate bugs to a single layer.

With this hard coded network definition in place and with hard-coded file links to the Caffe model text description for weights and bias and to the input image file, the example code is not very flexible for targetting different models. While most Caffe models should be targettable at the DPU, a bit of manual investigation will be required in most cases to evaluate suitability and so automatic network definition from Caffe files is not supported. For example larger networks might benefit from a larger DPU size (e.g. one that is built from 256, 512 or 1024 neurons).

The example code takes the hard coded network definition and builds up an image in memory to be transferred to the FPGA attached memory. The code loops through all enabled layers, up to 29, reading the weights from the Caffe file, and building the Instruction Word from the hard coded network information. Weight memory start addresses are dynamically allocated based on the memory used. Once the network image is completed, the input data is also added to the memory buffer

The example application then uses the Alpha Data ADXDMA driver and API to control the data transfer over PCIe to the ADA-SDEV-KIT3 board DDR3 memory. The DMA functions are used to simply copy the 12MB of data across to the DDR3 DIMM on the ADA-SDEV-KIT from where it can be accessed by the DPU.

The DPU is started with register writes to the start address. The CPU then simply polls the DPU registers every 250us until completion, displaying DPU state, time since the DPU start and time DPU active. By default this is just run once for all the DPU layers, but optionally each layer can be started in turn with the time for each layer recorded separately, which can be useful in determining the layer efficiency, which can be quite low for the batch

1 size used for certain layers with a high number of weights.

After the register reads detect that the DPU has competed, DMA functions then transfer back the data from the results areas of DDR3 into host memory. This data is then written out as two text files for further processing and analysis.

Deployment

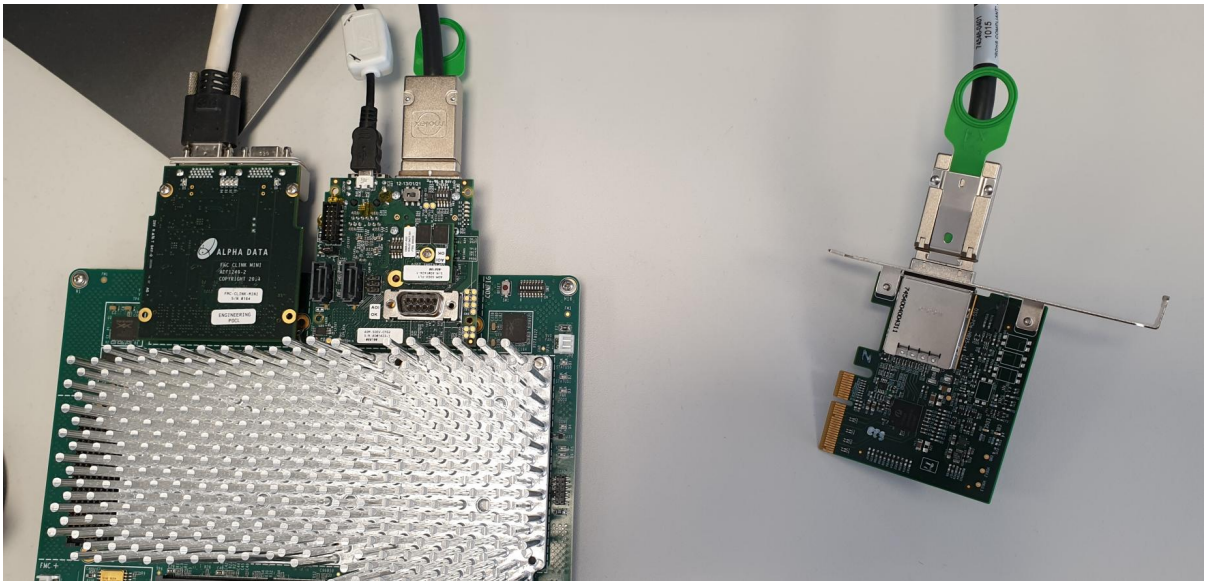


Figure 4 : ADA-SDEV-KIT3 Connection to PCIe

The equipment requied to run the application in hardware is the same as described in appendix A of *ad-ug-0142_v1_0 : ADA-SDEV-KIT DMA Demonstration FPGA Design Release: 1.0.0*. Specifically a Linux test machine is required to host the PCIe extender card. This machine (or another closely located) must also be able to facilitate the JTAG programming of the FPGA using Vivado Hardware Manager and a Xilinx USB JTAG cable. An ADA-SDEV-KIT3 (or ADA-SDEV-KIT2) powerd by an ATX power supply is required with the ADM-SDEV-CFG2 (or ADM-SDEV-CFG1) configuration FMC fitted to provide access to the IPASS cable. A PCIe extender card (e.g. OSS-PCIe-HIB25-x4-H) with an IPASS cable must be plugged into the test machine to allow the software to transfer data across to the FPGA and its attached DDR3 memory. Figure 4, shows the ADA-SDEV-KIT3 fitted with the ADM-SDEV-CFG2 module (and another unused FMC card) and also the far end PCIe card for connecting the other end of the IPASS cable.

Using this test set up the bitstream was tested by downloading over JTAG into the KU060 FPGA. The test Linux PC, running Ubuntu 18.04 was then reset allowing it to identify the PCIe endpoint. The ADXDMA driver was installed on the test Linux PC allowing it to identify and control the FPGA PCIe endpoint. The test application was compiled and run on this set up with the data and model files copied to the local directory. Testing was performed in both layer-by-layer mode and with a full end to end run.

Running a full end to end run results in a total DPU processing time of 38101671 clock cycles and an active time of 30166115 clock cycles. The clock used for the DPU is the 125MHz backend clock from the PCIe. In theory a faster clock could be used for the DPU and the AXI connections between the DPU and the DDR3 memory, improving the performace, but perhaps making the deisgn more difficult to follow. The resulting DPU processing time for the network is 304ms, operating at an efficiency of around 80%, with around 20% of the time being used to load weights. This efficiency could be improved by batching up the processing, assuming latency of response is not critical.

Conclusions and Summary

This paper series has covered the implementation of a DPU suitable for applications in radiation tolerant environments. The first paper covered a basic though rescalable DPU core implementation. The second paper covered the selective use of triple mode redundancy on the control side to make the design robust in those challenging environments. The third paper covered many of the practical data transport and control issues surrounding the core DPU processing, such as fetching the model weights and data from memory for the processing, and configuring and scheduling processing runs for each layer in sequence.

This final paper has taken the DPU design, proven in simulation used in the first 3 papers, and surrounded it with an appropriate shell design for deployment on real FPGA hardware. The ADA-SDEV-KIT3 is used as it is a reference platform for the KU060 FPGA (a part also available in a Radiation Tolerant package for Space flight deployment.) To ease the use of the core DPU IP with the other shell infrastructure, the VHDL code was packaged using Xilinx IP Packager. This then allows a simple TCL script to instantiate the board specific shell components (XDMA PCIe endpoint and DDR3 controller) along with the DPU and other AXI infrastucture in the same Vivado IP Integrator block diagram, which can then be used to generate the FPGA bitstream.

To test the bitstream a simple C application was written. This included as port of the VHDL testbench code which read in the Caffe Model layers from the text version of the Caffe Model file. This also includes a function for converting a model layer description into an instruction word for the DPU processor. The main function constructed a memory image to copy to the FPGA DDR3 SDRAM, and then used the ADXMDA driver and API to copy this across over PCIe. The application then triggered the DPU to run, and when the DPU completed, the application also copied back the results data over PCIe for further analysis, demonstrating the operation of the DPU on all the YoloV3 network layers and allowing some performance metrics to be read back.

This series has covered the implementation of a potentially radiation tolerant machine learning CNN solution on the Xilinx KU060 FPGA device. Starting from a flexible CNN implementation structure a general purpose DPU core was developed that can support many different neural network layers. The selective use of triple mode redundancy was employed to make the DPU design more robust to single event upsets. Higher level control and data transport structures have been wrapped around the processing core to step towards a practical implementation and a few other network specific ad-hoc features were added to specifically support the YoloV3 structure. This core was then targetted at the KU060 device on the ADA-SDEV-KIT3 development board, and wrapped up with support for DDR3 external memory, external PCIe population and readback of the DDR3 with the model parameters and network data, and minimal PCIe top level control to start and monitor the DPU progress from host CPU software. In general, however this DPU requires very little external CPU interaction, and therefore could be embedded in a different FPGA system, where the data is collected directly via another interface, running of the DPU is automatically triggered by this, and results are acted upon by other hardware modules in the FPGA. And therefore the DPU could be used in situations where external CPU control is not available.

Revision History

Date	Revision	Nature of Change
20/01/22	0.1	First draft
21/03/22	1.0	First 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