Breaking Memory Bandwidth Barriers using HBM FPGA

Breaking Memory Bandwidth Barriers using High Bandwidth Memory FPGA

A. C. McCormick, Ph.D., Technical Director, Alpha Data Parallel Systems Ltd.

Introduction

The release of Virtex Ultrascale+ High Bandwidth Memory(HBM) FPGA devices, opens up whole new areas of memory bound applications to the benefit of power efficient FPGA acceleration. A recent increasing trend has been to target a variety of memory bound applications to GPU systems, simply because of their significant memory bandwidth advantage over the CPU, and this is despite the application not having any need for the GPUs primary functionality: the very high performance parallel floating point arithmetic. With the advent of FPGAs with similar external memory bandwidth, but much more flexible and higher internal memory bandwidth configurability, more customized and energy efficient accelerated solutions for these problems are now possible.

The VU37P is the largest device in the Xilinx Virtex Ultrascale+ HBM range. This device uses Xilinx 3D Stacked Silicon Interconnect to stack multiple FPGA dies, including one with a very high bandwidth memory controller along with two 4GB HBM Gen2 DRAM dies into the same package, allowing massive bandwidth between in-package wafers. This coupling of massive parallel processing capability and massive memory bandwidth within a single device could result in orders or magnitude acceleration in traditionally memory bound applications.

The Alpha Data ADM-PCIE-9H7 is the first Virtex Ultrascale+ VU37P board in the market place. This board provides the VU37P FPGA, with 2.8 Million configurable logic cells, 60MB of very flexible on-chip (cache) memory, 9024 DSP tiles (potentially over 500 GFLOPs double precision performance), 8GB of on-package HBM memory with 460 GB/s memory bandwidth, Gen3x16 PCIe connectivity with host memory (or dual OpenCAPI 25Gx8), and another 48x 25Gb/s links that are available to connect to other FPGA boards, or a 100G Ethernet network.

In this white paper, 3 case studies are investigated to assess the potential performance of this board with real world applications: multi-dimensional FFTs, Merge Sort and Matrix Multiplication.

Multi-Dimensional FFT Implementations

The FPGA is the device of choice for FFT implementations in many aerospace and defence systems, implementing real time radars, sonars and communication systems. This is due to the high efficiency of implementations which can exploit the extremely high bandwidth of dual ported on-chip memories, to buffer data between FFT butterfly operation engines, which themselves can be easily customized to an appropriate bit width for the application. Single dimensional FFT performance in larger devices can be limited by the IO bandwidth. Multi-dimensional FFTs can use the data out at each stage, and the HBM architecture may be very well suited for this task as results can be stored and corner turned efficiently, staying within the chip package.

FPGAs allow very efficient architectures for FFT computations to be built. The specialized multiplication logic (DSP tiles) provide very efficient multiply accumulate hardware for implementing a Radix-2 or Radix-4 Butterfly cores for each stage of the FFT. Between stages, the dual port block RAMs provide buffers that can be simultaneously read from and written to. This allows a pipelined implementation, that can continually run and

push data through all the stages, fully utilizing the multiplication hardware. Figure 1 shows the general structure of a pipelined FFT as implemented in an FPGA.

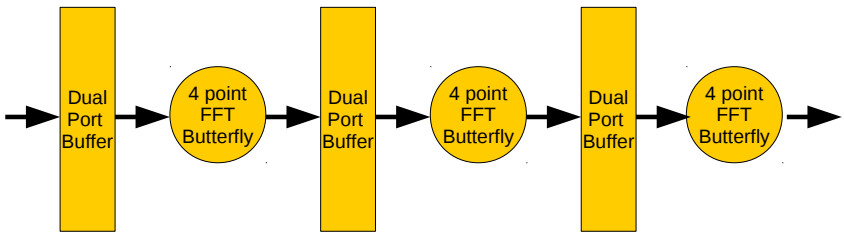

Figure 1 : Pipelined FPGA FFT Implementation

Efficient implementations are available off the shelf for FPGA designs using the IP catalogs available in most FPGA design tools. Therefore is no need to implement this from scratch unless there are very specific requirements to be met. The Xilinx IP Catalog in Vivado, provides fixed and single precision floating point implementations, that can run at clock rates in excess of 400MHz. These can be configured to a range of sizes and structures. This paper will focus on an 8192 point implementation, as as 8192x8192x8192 complex single precision 3D FFT will occupy 4GB of HBM RAM, allowing for efficient double buffering in the 8GB VU37P part. The Xilinx FFT IP core can provide a pipelined implementation of a single precision 8k FFT, with 1 sample per clock cycle performance, using around 100 DSP tiles, with effective performance of 26 single precision GFLOPS. As this occupies only a small fraction of the FPGA, for the 3D FFT accelerator design, a large number of these will be implemented in parallel, and this will also help efficiently implement the transfer of data from the HBM memory to and from these cores.

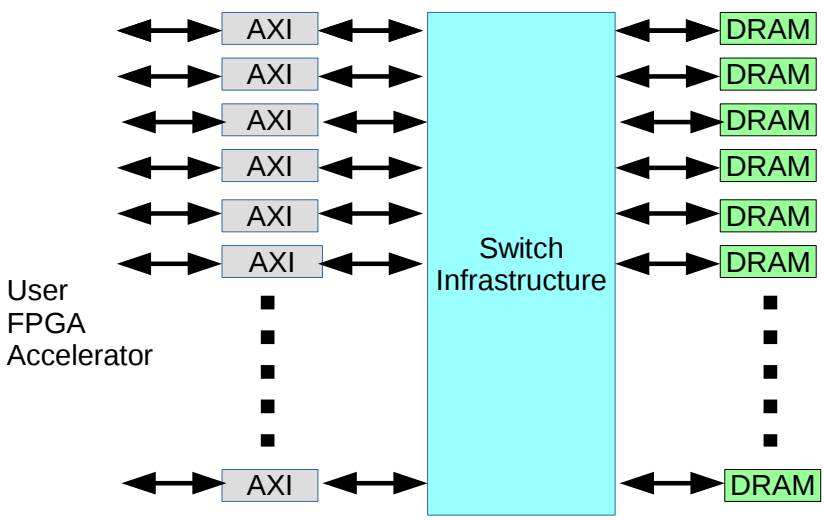

Figure 2 : High Bandwidth Memory Access Structure

A parallel implementation within the chip is essential to fully exploit both the procesing performance and memory bandwidth available. The HBM memory is still DDR4 based and therefore is best accessed via long contiguous bursts, and performs poorly when accessed randomly and with data accesses less than the port width. The memory also consists of 32 independent parallel 256 bit wide AXI ports, each which can access the entire HBM address space through switch logic in the chip. The HBM memory is also a parallel stack of DRAM devices, and so accesses of at least 256 bits wide will be required for reasonable efficiency. Figure 2 shows the general HBM switching structure.

The ratio of memory access to computational performance is critical here. A single FFT core, will require a sustained 3.2 GB/s read performance to match the 26 GFLOPs performance. The same bandwidth is also required to write back the processed data. A single AXI port will provide at best four times this read bandwidth, and assuming read and write back each requiring a port, the device could potentially support up to 64 cores in parallel, based on memory bandwidth considerations.

With multi-dimensional FFTs, the data access pattern may not necessary fit how the data is stored in memory. A 3D FFT involves performing FFTs in each dimension in turn. Reading the FFT input in the first pass, the X direction will be efficient, but reading data for the Y and Z direction FFTs directly from DRAM requires inefficient 64 bit wide, burst length 1, transfers, which will waste at least 75% of the bandwidth. The solution is to perform corner turning on the data, in the internal RAM within the FPGA. This can be performed, while reading from memory, during the FFT write back, or between FFT transfers, using a separate accelerator core. The first option is considered in this paper.

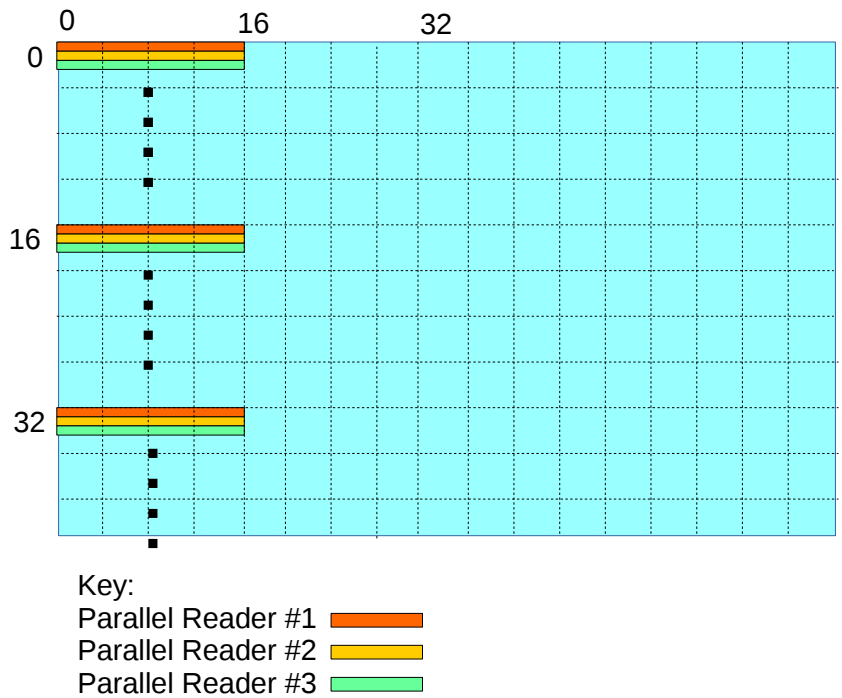

Figure 3 : Corner Turning FFT Reads from Memory

In this case study, each parallel FFT core is paired up with a memory reader and memory writer engine. The memory readers first push their data through a corner turn block which aggregates data from all the parallel readers and outputs it after a corner turn transform. For the first FFT pass in the X direction, this corner turn can be bypassed. Figure 3 shows the memory access pattern for 16 parallel cores using the corner turn. The first unit reads a slice, 16 channels wide, from the first line (shown in orange). The next read by this DMA reading engine is then from the 17th line, and the 33rd line etc. The second DMA engine reads from lines 2,18,34, the third from 3,19,35 etc. The corner turn block receives all 16 parallel input streams and outputs them as 16 parallel corner turned streams, where each FFT core gets 1 sample from each of the 16 readers for each 16 lines in the Y direction (in the Z direction, the same logic works with a different address jump between lines.)

The choice of FFT size of 8192 has been motivated in this case by the match between the HBM memory and the footprint of an 8192x8192x8192 data set. The corner turning approach is suitable for a range of larger FFT sizes, with the memory bandwidth effectively limiting the number of parallel cores to 64, smaller FFT sizes than 4k will be less efficient. With significantly smaller FFT sizes, such as 128x128x128, the corner turning may be more effectively performed between FFT stages, with on-chip SRAM memory within the FPGA, and therefore the use of HBM may be less important in these applications.

When implementing this solution, the limiting resource turns out to be the availability of BlockRAM blocks used as the double buffers within the FFT cores. The VU37P has a very large on-chip SRAM memory, but 80% of this is larger UltraRAM blocks, which are not currently used by the FFT IP Core. This limits the number of FFT cores to 48, however this still results in a 3D FFT core capable of sustained operation in excess of 0.75TFLOPs (single precision) using a conservative 250Mhz system clock.

Parallel Merge Sort

Sorting and searching algorithms are fundamental building blocks of many applications. Without any floating point arithmetic requirement, they are often ignored as acceleration candidates. Research into optimizing these algorithms often only focuses on minimising the number of comparison operations, through some heuristic means. With FPGA implementation, it is however clear that the comparison operation, is actually not a significant part of the processing cost, and processing time. Much more performance and energy is used in moving the data from memory to the comparison unit and back. Therefore an HBM enabled FPGA with massive bandwidth between comparison logic and the memory should be able to provide a better solution. The Ultra RAM and Block RAM within the FPGA which allow very application specific control of data caching and data flow will also aid in providing higher performance and lower energy solutions.

Merge Sort is not an algorithm typically considered for FPGA implementation. However for large data sets, it is a deterministic, efficient sorting algorithm, that can be easily parallelized. The arithmetic requirements are minimal. It can still benefit from the flexible configuration of the on-chip memory to provide a very efficient low power solution. Since computation is minimal, the movement of data dominates the performance and power requirements of this algorithm, and therefore there may be substantial benefits to using HBM with this algorithm.

Merge Sort is a divide and conquer approach to sorting data. It has a deterministic complexity $O(N \log_2 N)$, which may be slower than some heuristic algorithms with best case data (e.g. Quick Sort), but it does not suffer from data dependent issues. It is also parallelizable, making it better suited for FPGA implementation.

The $O(N \log_2 N)$ complexity is similar to that of the FFT algorithm, and a similar data flow structure as described in the previous case study can be employed to create a $\log_2 N$ parallelism, through a pipeline, which will make the sort time O(N), effectively matching the rate at which data can be read from memory, and written back.

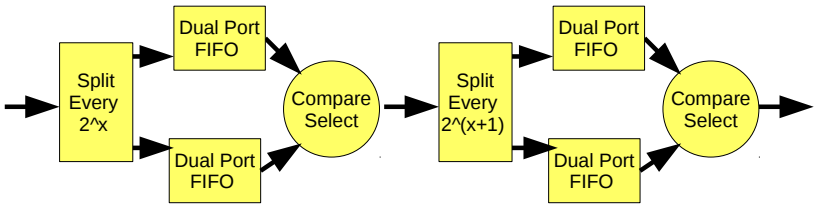

Figure 4 : Merge Sort Pipeline Stages

Figure 4 shows 2 stages of the merge sort pipeline. At each stage the input stream is split into 2 FIFOs, and the comparison operation is used to select the larger output first to push to the next stage. The input data stream is fed alternativeley to one of 2 FIFOs. At the first stage, data is unsorted and so alternative elements are sent to each FIFO. In later stages, input data will be sorted into 2^x long sections, and so the split occurs after this number of elements. At each stage once 2^x elements have been read from one FIFO, the remaining data will be read from the other FIFO until $2^{(x+1)}$ sorted elements have been output.

The FPGA resource requirement for the comparison and select operation is relatively minimal and requires just a handful of logic cells (considering 2.8 Million are available). For maximum performance the comparison operation is hard coded, in the example case to a 64 bit big endian comparison (text character order), however more complex comparisons (e.g. 64 bit little endian for integer, or even case insensitive ascii byte by byte comparison) can also be easily implemented. Run time, software selectable comparison units could also be added at only a small extra logic cost (effectively implementing all possible comparisons that could be required). Data width is less flexible, and the data elements need to be divided into values that are used for comparison and other data elements, to be carried along in the sort. In the example case study, an 8 byte value was paired with an 8 byte key, not used in the comparison, but which could be a 64-bit pointer into the reference database.

The components which take up the resources in this accelerator are the FIFOs. At each stage, the FIFO needs to be 2^x elements deep. This effectively puts a limit on how many stages can fit in the FPGA. Beyond this, the HBM device allows additional stages to merge sort from one port on the HBM device to another.

Using the VU37P device and choosing 16-byte wide data records (8 bytes of comparable value, 8 bytes of key/ pointer) it is possible, building the FIFOs out of Distributed, Block and Ultra RAMs to construct 20 parallel stages, alllowing a parallel sort of over 1 million elements (16MB) in O(N) time, at the memory port read rate. Using the other HBM ports to add additional stages allows an extra 8 sorts to be placed in the pipeline, providing the potential to sort 4GB of data in O(N) time.

Double Precision Matrix Multiply

Linear algebra libraries are at the heart of many HPC applications, and matrix multiplication is one of the most commonly used operations. Matrix Multiplication is $O(N^3)$ in its naive implementation, whereas the memory bandwidth requirement is $O(N^2)$. FPGAs can efficiently implement fixed sized **N** units where the data for a row and column for each multiplier unit can be cached local to that unit. The local memory and processing resources will limit the size of **N**: larger values will help overcome any memory bandwidth limitation, however smaller values may be more flexible and support a wider number of matrix sizes more efficiently. HBM devices can allow better exploitation of FPGA Matrix Multiplication cores, as typically the fixed multiplication size needs combined with simpler memory bound operations such as addition, transposition and scaling within an iterative loop, and the HBM bandwidth and multiport structure will allow these additional operations to operate on the same data, and keep the dataset locally on the device.

While there are some matrix multiplication algorithms that can perform better than $O(N^3)$ these either have stability issues for certain data sets, or have other computation complexities that make them unsuitable in most practical use cases. Therefore most algorithms tend to still use the naive $O(N^3)$ implementation. This is not necessarily memory bound as for the $O(N^3)$ computations only $O(N^2)$ memory accesses are required. Therefore the ratio of computation to memory access is O(N) and therefore to avoid memory bound issues it would appear that increasing N will improve the situation. However to achieve this, the appropriate row and column data must be cached close to each multiplication unit. In CPU systems, this can result in very fast performance with small matrix sizes, dropping to poor performance with larger matrix sizes, dominated by cache misses.

With FPGA implementations, the dual port memories, allow a double buffered cache implementation, where the next Matrix operation data can be loaded, while the current matrix computation progresses. Since there is a O (N) factor of data re-use within the algorithm, the writes from memory to the caches will take far less time than the computation, avoiding any cache dependency. An FPGA matrix multiply core can consist of N floating point multiply units, each with two local cache memories contaning a column from the first matrix and a row from the second. Iterating across the multiplication of these two vectors will produce a partial product for each of the NxN results, which can be summed with all the products from the other multiply units using an adder tree pipeline with N-1 douple precision floating point adders. Figure 5 shows this systolic array structure for matrix multiplication. The latency from cache line read to matrix product element output is quite long, consisting of the double precision multiply latency plus $\log_2(N)$ times the addition latency. However relative the the N^2 multiplications, this time will be small, and the next matrix multiplication can be started immediately after the last finishes reading data, and so the latency may not be critical to performance.

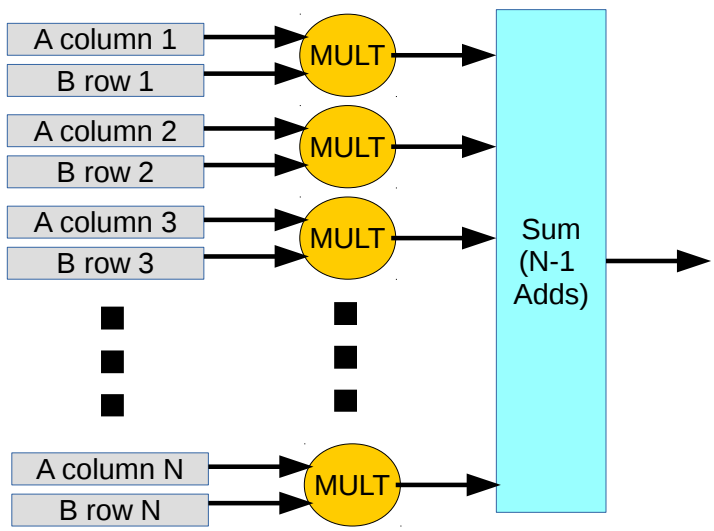

Figure 5 : Matrix Multiply Systolic Array

The local caches will be implemented in the FPGA using Block RAM or Ultra RAM. This places a size restriction on these blocks, making them a power of 2 size. For non-power of 2 parallelizations of N, this does introduce a memory use inefficiency. To double buffer the data, and allow the data load from external memory to run in parallel with the matrix operation on the previous data frame, the caches also need to be at least 2N in size. Block RAM components naturally map to 512 element deep memories, and therefore supporting a parallelization of up to 256 units. Moving from 256 to 512, not only increases the RAM requirement in proportion to the parallelization N, but also requires a doubling of each local buffer size, as does moving beyond a parallelization

of 512. With this restriction in mind, the matrix multiply core used was scaled up to fill as much of the VU37P as possible. Using N of 512 required less than 50% of the computational resources but more than 50% of the RAM. Scaling up to a parallelization of 1024 does not appear possible, but a parallel 704x704 double precision matrix multiplier does fit. If clocked at 250MHz, this would have a performance of around 350 GFLOPs.

The matrix multiplication implementation as described so far would work equally well on a non-HBM FPGA as the memory bandwidth is not the limitation. The advantage of using HBM starts to show when combining the matrix multiplication operation, with other matrix operations. Since the Matrix Multiply core reads from 2 matrices and writes back to one, it can use at most 3 of the HBM AXI ports to access the memory. This leaves many other ports free to perform complimentary computations in parallel. One useful operation, is matrix element by element addition. This can be used as part of a divide and conquer algorithm for multiplying larger matrices, and could allow the core to handle much larger matrices that its fixed size. The core can also handle smaller matrices, by setting unused rows and columns to zero. However this is relatively inefficient as the computation time will take the same length of time as for the full matrix size.

Future work will look into the advantages and disadvantages of using multiple smaller cores in place of the single large matrix multiply core, which can be easily implemented in HBM parts due to the multi-port access to the HBM. This could be combined with not only the extra matrix addition operation, but several other matrix operations including Matrix-Scalar multiplication, Matrix-Vector multiplication, Matrix Transposition and element by element recipricol or square root, which are often used in HPC application loops along with a large Matrix-Matrix product operation. Some of these operations can be pipelined together reducing the memory requirement, for example computing the square root of the matric product before its written back to memory.

Conclusions

This white paper has discussed the implementation of 3 application case studies on the Xilinx Virtex Ultrascale+ HBM VU37P device, running on the Alpha Data ADM-PCIE-9H7 accelerator card.

While conventional FPGAs provide a very efficient mechanism for FFT implementation, for large multi-dimensional workloads, the high bandwidth parallel HBM interface provides the perfect buffer for handling these larger data sets, especially when combined with the FPGAs on board memory flexibility to allow efficient corner turning of data.

Parallel Merge Sort can also be targetted at the HBM FPGA device. The large FPGA size of the VU37P, and especially the large provision of Ultra RAM allows a very efficient implementation of sort pipeline with up to 20 parallel stages, sorting over a million elements as fast as they can be read from memory. The HBM allows additional parallel sort stages to operate in the pipeline, extending the sort size by several orders of magnitude.

Large parallel Matrix Multiplication cores can be implemented on most large FPGAs, and these implementations are not specifically memory bandwidth sensitive. However HBM allows easy combination of one or more Matrix Multiplication cores with, more basic lower performance, memory bound operators such as addition or scaling, that allow the data set to be kept on the accelerator while perfoming a more complex processing loop. This multi-port parallel access to the same working data set, which can be up to 8GB in size will enable many combinations of accelerators to be implemented in the same FPGA design - for example, combinations of Matrix-Multiplication and FFT processing have many applications.

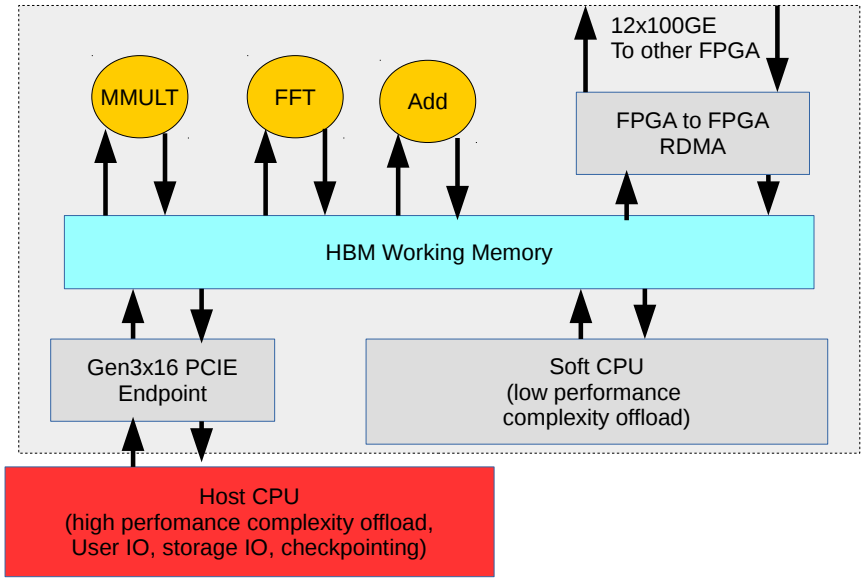

Figure 6 : Heterogeneous Compute Acceleration Node

This architectural idea of a compute acceleration node with multiple heterogeneous acceleration cores can be expanded to an almost generic memory and dataflow centric paradigm, where the working data for the application is kept in the HBM memory and the processing operates around this. Figure 6 illustrates one such structure. The FPGA contains a number of different accelerators and work modules. Simple and complex computation accelerators operate directly on the HBM memory. Control and mode complex tasks can be handled by either an on-chip soft CPU, or by a host CPU over PCIe. Internode communication can operate in parallel, shifting data through the 1.2TB/s of IO bandwidth between the different FPGAs in the cluster.

Case Study Vivado Projects

The case study Vivado projects are available for customers of the ADM-PCIE-9H7 HBM FPGA accelerator card. Please contact Alpha Data for information on how to access these.

Address: 4 West Silvermills Lane, Edinburgh, EH3 5BD, UK
Telephone: +44 131 558 2600
Fax: +44 131 558 2700
email: sales@alpha-data.com
website: http://www.alpha-data.com

Address: 611 Corporate Circle Suite H Golden, CO 80401
Telephone: (303) 954 8768
Fax: (866) 820 9956 - toll free
email: sales@alpha-data.com
website: http://www.alpha-data.com