Evaluation of ‘OpenCL for FPGA’ for Data Acquisition and Acceleration in High Energy Physics

Srikanth Sridharan[1]
Marie Curie Fellow, CERN,
CH-1211 Geneva 23, Switzerland

E-mail: srikanth.sridharan@cern.ch

Abstract. The increase in the data acquisition and processing needs of High Energy Physics experiments has made it more essential to use FPGAs to meet those needs. However harnessing the capabilities of the FPGAs has been hard for anyone but expert FPGA developers. The arrival of OpenCL with the two major FPGA vendors supporting it, offers an easy software-based approach to taking advantage of FPGAs in applications such as High Energy Physics. OpenCL is a language for using heterogeneous architectures in order to accelerate applications. However, FPGAs are capable of far more than acceleration, hence it is interesting to explore if OpenCL can be used to take advantage of FPGAs for more generic applications. To answer these questions, especially in the context of High Energy Physics, two applications, a DAQ module and an acceleration workload, were tested for implementation with OpenCL on FPGAs[2]. The challenges on using OpenCL for a DAQ application and their solutions, together with the performance of the OpenCL based acceleration are discussed. Many of the design elements needed to realize a DAQ system in OpenCL already exists, mostly as FPGA vendor extensions, but a small number of elements were found to be missing. For acceleration of OpenCL applications, using FPGAs has become as easy as using GPUs. OpenCL has the potential for a massive gain in productivity and ease of use enabling non FPGA experts to design, debug and maintain the code. Also, FPGA power consumption is much lower than other implementations. This paper describes one of the first attempts to explore the use of OpenCL for applications outside the acceleration workloads.

1. Introduction

The proposed upgrade for the Large Hadron Collider LHCb experiment at CERN envisages a system of 500 data sources, each generating data at 100 Gbps. The acquisition and processing of this is a challenge even for state-of-the-art FPGAs. This challenge has two parts, one is Data Acquisition (DAQ) and the other Algorithm Acceleration, the latter not necessarily immediately following the former.

For the Data Acquisition, a Header Generator module was needed to packetize the streaming data coming in from the front-end electronics of the detectors, for easy access and processing by the servers. This necessitates FPGA architectures that not only handle the data generated by the

[1] This work was done under ICE-DIP, a European Industrial Doctorate project funded by the European Community’s 7th Framework programme Marie Curie Actions under grant PITN-GA-2012-316596.

[2] Altera OpenCL Compiler version 14.1 was used for this work.

experiment in real-time, but also dynamically adapt to potential inadequacies of other components, such as the network and PCs, while ensuring system stability and overall data integrity. Since the data source has no flow control, this module needs to modify the stream data by dropping datasets in a controlled fashion if a back pressure signal is generated by the downstream modules. A front-end source emulator capable of generating the various data patterns that can act as a test bed to validate the functionality and performance of the Header Generator was also needed. Such a system was earlier designed and realized in VHDL. [1]

While this process has been traditionally carried out using hardware description languages (HDLs), the possibility exists of using OpenCL to design a DAQ system. OpenCL has the potential to simplify the development cycle of the applications, and make it easier for physicists, who are more familiar with traditional software, to understand the system and make modifications in the future. This is challenging due to the fact that the OpenCL language is designed for Parallel Processing and not really targeted at real-time DAQ and there are major challenges in representing the cycle-accurate data acquisition and processing system in OpenCL. Still, OpenCL for FPGAs may be applicable from a high level synthesis perspective. Achieving this will enable the transition of the entire FPGA design flow for High Energy Physics applications to OpenCL, rather than just the algorithm acceleration portion that involves parallel processing.

For the algorithm acceleration part, the Hough transform algorithm [2] was implemented in OpenCL. This is a method to identify patterns from points in 2D/3D space and can be used to identify particle tracks from hits in the VELO detector elements. Variations of this algorithm are also used for feature identification on the data from other detectors. This work explored the feasibility of implementing Data Acquisition and processing system on OpenCL and evaluated the performance of this OpenCL accelerated application on FPGA and a GPU. Development used the Altera OpenCL compiler for FPGA.

2. FPGAs in High Energy Physics

FPGAs are used in HEP experimental setups for a variety of purposes. They are used initially for Data Acquisition to collate the streaming data coming off the front end electronics over multiple channels. FPGAs can also be used in the low level trigger system where the acquired data need to be quickly processed to arrive at the trigger decisions. The custom nature of the solutions required and also the need to operate in high radiation environments make any other technology unsuitable for these purposes. ASICs are suitable only for high volume production and are unviable for these applications due to prohibitive costs.

2.1. HDL based FPGA Design

The custom circuitries needed to implement these systems are traditionally designed using Hardware Description Languages like VHDL/Verilog. Programming in VHDL/Verilog is done at the Register Transfer Level (RTL) abstraction. These designs are then synthesized into netlists that are then placed and routed for the specific FPGA device and finally a bitfile is generated to program the FPGA. VHDL/Verilog being niche languages used by FPGA/ASIC designers, knowledge of them is not common among physicists or even software engineers. The RTL abstraction is a very low level representation and hence it is also difficult to design and debug extremely large designs. Moreover implementing the necessary logic/function in HDL is not enough - one needs to manually create memory hierarchies and also instantiate communication cores and link them to the design to keep the system supplied with data. All the control and glue logic needed to keep the design in synchronicity with other systems also need to be implemented. This just models the FPGA side of the system; In addition, the PC control software still needs to be implemented. Despite the shortcomings, it is nevertheless very flexible and versatile enough to implement anything from a custom processor, to a DAQ system or a co-processor/accelerator.

2.2. OpenCL based acceleration on FPGA
The Khronos Group [3], the maintainer of the OpenCL specification, defines it as an open standard for parallel programming of heterogeneous systems. It is the first open, royalty-free standard for cross-platform, parallel programming of modern processors found in personal computers, servers and handheld/embedded devices. OpenCL has been in use for a while to take advantage of GPUs, DSPs and Manycore processors for parallelizable workloads. While FPGAs have always been capable of exploiting parallelism, their hardware based programming model makes taking advantage of them harder than using GPUs and other devices. That has now changed with the two major FPGA vendors supporting OpenCL-based acceleration by means of SDKs. These reduce OpenCL kernels to custom circuits that are subsequently synthesized to netlist and a bitstream is generated. The real advantage of OpenCL-based acceleration on FPGA is that the user only needs to focus on describing the logic for the computation itself. All the additional control logic, the PCIe core for communication and data transfer to and from the PC and the necessary memory hierarchy plus the memory controllers, are automatically generated by the tool. For instance, the tasks of transferring data to the FPGA, executing the kernel, and then retrieving the results is reduced to 3 simple API calls as follows:

```
Copy Data Host → FPGA: clEnqueueWriteBuffer( ... );
Execute Kernel on FPGA:  clEnqueueTask( ..., my_kernel, ...);
                         clEnqueueNDRangeKernel( ..., my_kernel, ...);
Copy Data FPGA → Host: clEnqueueReadBuffer( ... );
```

This makes exploiting FPGAs for acceleration as easy as using GPUs. With the possibility that OpenCL can be used to leverage FPGA for acceleration, this opens the possibility that more can be achieved. Could traditional FPGA designs, for example of DAQ, be implemented using OpenCL?

3. Implementing a DAQ system on FPGA with OpenCL
In order to evaluate the possibility of implementing a DAQ system using OpenCL, an existing design was used. The Adaptive Header Generator [1] is an existing module in DAQ flow for the LHCb experiment. Its purpose is to packetize the streamed data by creating a meta-header by combining the header information of hundreds of small event datasets. The Source Emulator creates a pseudo data stream. It serves the dual purpose of being a synthesizable test bench for the Header Generator and at the same time can function as a standalone module that can be integrated into other systems if required. This is not a parallelizable problem. Therefore it is an ideal test case for exploring OpenCL for non acceleration applications on FPGAs.

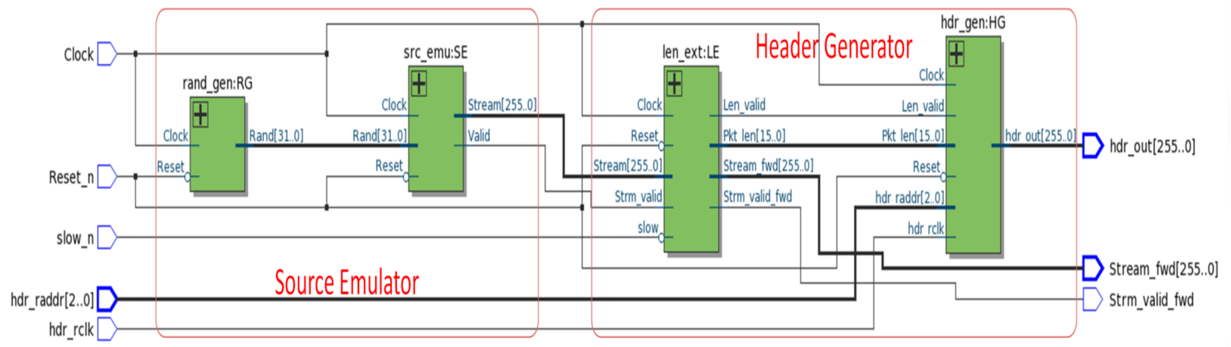

Figure 1. Complete data flow from random seed for source emulator to the generation of the modified data stream and the header.

The existing Header Generator and the Source Emulator modules were implemented in VHDL. Figure 1 shows the data flow of the DAQ system with the major sub modules. The functionality of these modules was captured in OpenCL kernels and implemented using Altera's OpenCL compiler. The following sections describe the challenges faced in implementing this and their solutions.

3.1. Handling IO and moving data between kernels

The very first task in DAQ is to clock the data into the FPGA. The OpenCL standard was initially created in order to exploit the parallelism of the many cores found in devices like GPUs and those devices are only capable of accelerating parallelizable workloads. Since the original OpenCL spec does not have a provision for a generic I/O mechanism outside of communication with the Host CPU, an additional talk is to communicate the data between various kernels. In the OpenCL specification, again keeping in mind devices like GPUs, the only way to move data was by means of global memory access. The latencies involved with memory access would be too high for DAQ applications with streaming inputs. To overcome these limitations and meet these requirements, channel extension as provided by Altera can be used. Channels are FIFO based structures that can be used to move data between:

- I/O → Kernel
- Kernel → Kernel (bypassing the Global Memory)
- Kernel → I/O

Using channels, the movement of data signals such as Rand and Stream, as shown in figure 1, were modelled. The concept of channels is incorporated in the next OpenCL 2.0 specification as pipes.

3.2. Bit level manipulation and non standard data width

A need for bit level manipulation and variables with non-standard widths is common with Data Acquisition systems. In the Header Generator system, it was a needed to access a 16 bit ushort variable in chunks of 11 and 5 bits as well as whole to implement an addressing logic as shown in figure 2.

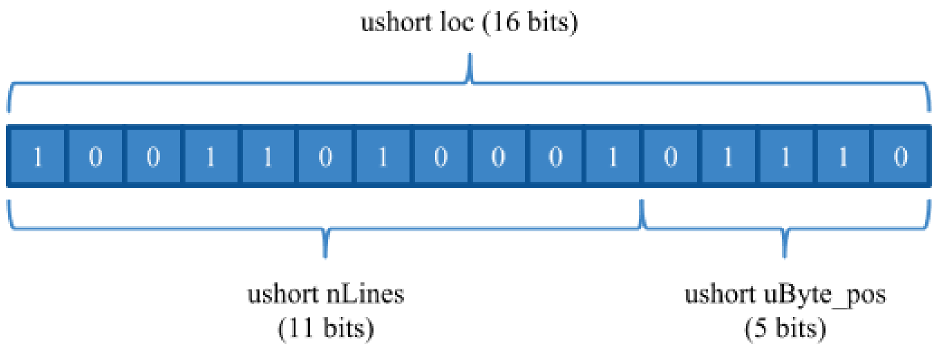

Figure 2. Accessing a memory element in parts and in whole.

This type of memory access is not possible in devices like GPUs and has to be accomplished in an indirect manner with CPUs. However, this is trivially accomplished on an FPGA as follows:

```
SIGNAL location : STD_LOGIC_VECTOR(15 DOWNTO 0) ;
alias uByte_pos : STD_LOGIC_VECTOR(4 DOWNTO 0) is location (4 downto 0);
alias nLines : STD_LOGIC_VECTOR(10 DOWNTO 0) is location(15 downto 5);
```

This kind of packed representation can be accomplished by means of bit fields. This feature, although present in the C, is absent from the OpenCL specification. Luckily, Altera's OpenCL compiler supports bit fields as it makes perfect sense on an FPGA. This can be implemented with OpenCL as follows:

```
typedef union loc
{
    Struct
    {
        uchar uByte_pos :5;
        ushort nLines   :11;
    };
    ushort location;
};
```

This not only makes writing and understanding the code easier, but also results in efficient hardware on the FPGA.

3.3. Control Signals

Control signals are integral to any Data Acquisition system. In our design from Figure 1, apart from the data signals that we modelled as channels, these are the remaining signals:

- Clock and Reset
- Different Valid Signals (Strm_valid, Len_valid, Strm_fwd_valid)
- Throttle/Feedback signals (slow/slow_n)

3.3.1. Clock and Reset Signals. Altera's OpenCL compiler internally takes care of the clocking and reset of the kernel logic as well as for the memory access and communication with the Host CPU. These are hidden from the user.

3.3.2. Valid Signals. In the HDL implementation of the design, valid signals corresponding to the different data signals need to be explicitly created. However when the data signals are modeled as channels with the Altera's OpenCL extension, the compiler automatically creates a valid and stall signal corresponding to each channel. Hence these are also hidden from the user.

3.3.3. Throttle/Feedback signals. In DAQ systems dealing with streamed data, feedback signals are employed to convey congestion information from downstream to upstream modules. Using these signals, the upstream modules can stop or slow the data flow so as not to overflow the buffers. These signals are even more critical in systems such as the Header Generator where its own data source does not have any flow control but its data sink requires flow control. The 'slow' signal in Figure 1 serves this purpose and is generated by modules downstream of the Header Generator. When the slow signal is active, the logic in the Header Generator drops events from the data stream in a controlled fashion. The feedback signals are by definition asynchronous to the system and their relevance is instantaneous in nature. Unlike data signals where every single datum transmitted needs to be captured and in the same order, in the case of the throttle signal, only the current value is needed and it is critical that the current value is always available. There is no mechanism to model these signals in OpenCL. The usage of channels in this scenario was considered but since channels operate in a synchronous manner and the writes and reads to a channel always need to be in a 1:1 ratio, it is not suitable. Moreover, the internal flow control of the channels would interfere with the custom flow control of the system. Since the throttle signal cannot be implemented currently in OpenCL, this has become a road block in the attempt to realize a DAQ system on FPGA with OpenCL.

3.4. Conclusion: DAQ Implementation with OpenCL

To the author's knowledge, this is one of the first instances to explore using OpenCL for applications outside of acceleration workloads. Acceleration workload in this context will be something that is parallelizable to complete the computation faster. This is in contrast to a monolithic non-acceleration workload such as DAQ. It may have sub-modules that are pipelined, but beyond that there is little parallelization involved and nothing to be accelerated. While the current OpenCL specification and the Altera specific extensions do not support this kind of usage, it is not impossible to add proper support. Allowing asynchronous channels and adding support for 'VHDL signals' or 'Verilog wires' as FPGA specific extensions to OpenCL will facilitate modelling of these kinds of requirements. Since the underlying hardware is an FPGA that supports these kinds of operations and also since the Altera compiler actually generates a Verilog representation of the system from the High Level OpenCL constructs, it should be straight forward to add these functionalities. Altera has been notified both of this inadequacy in their current compiler and its possible solution described above. It is also possible that there might be more features and functionalities required by DAQ applications that were not encountered in this test application.

Since this is a novel attempt, the performance of the DAQ system is hard to predict. There are various factors that affect the prior estimation of performance of such as a system. Since the coding is done in OpenCL, that is converted to Verilog and then synthesized for FPGA by Altera's tools, the internal structures of the kernels are hidden from the user. There will also certainly be an overhead involved with the OpenCL-based design. Finally, unlike custom-designed memory interfaces, the use of an auto-generated memory hierarchy in OpenCL can introduce uncertainties in memory access. All of these can make it hard to estimate the performance. Nevertheless the ease of use and the massive gain in productivity this approach offers outweigh the limitations and warrant further investigation on these lines.

Table 1. Run times of kernels in ns

	FPGA (Altera Stratix V)			GPU (Nvidia GeForce GTX 680)		
	Kernel 1 (XZ)	Kernel 1 (YZ)	Kernel 2 (Overlap)	Kernel 1 (XZ)	Kernel 1 (YZ)	Kernel 2 (Overlap)
Run 1	3570722	3512339	8052825491	7980800	7975776	17914624
Run 2	3554957	3509823	8052820252	7980352	7982176	17966112
Run 3	3556479	3570346	8052903192	7981856	7974848	17892672
Average	3560719.33	3530836	8052849645	7981002.67	7977600	17924469.33

4. Algorithm Acceleration on FPGA with OpenCL: Hough Transformation

At the same time as exploring the possibility of using OpenCL for implementing a DAQ system, the use of OpenCL for its intended purpose, acceleration, was also investigated. One of the claimed advantages of OpenCL is portability of code. Hence, for this evaluation, an existing implementation of the Hough Transform code for a GPU was used [4]. The Hough transform is a method to identify patterns from points in 2D/3D space. Variations of this algorithm are used to identify particle tracks from detector data of various experiments at CERN. The version used with the VELO (Vertex Locator) Detector for identifying straight line tracks was used for this evaluation. The existing kernels were used and minimal changes were done to the host code to allow for precompiled binaries as Altera's OpenCL compiler only supports offline compilation. The code consisted of two kernels. The first was for a 2D Hough transform which is executed twice, for each of the XZ and YZ projections of the 3D particle hit data; the second was to find the overlap between the results of the two 2D transforms to compute 3D tracks.

4.1. Discussion of Results

The runtime figures of the two kernels over multiple runs are provided in Table 1. These runtimes were obtained by profiling for the start and finish of each kernel instance. These numbers are reported based on the internal clocks of the devices and hence the resolution of these values could be different. Also the total runtime of the program (host code plus kernels) is not shown due to the difference in the host system configurations. This also avoids the error due to the difference in online compilation (GPU) and offline compilation (FPGA). At the outset, the FPGA performs about 2.25 times faster than the GPU for kernel 1. But the runtime for kernel 2 on FPGA is abnormally high and hence the GPU is faster for kernel 2 by two orders of magnitude. As already mentioned, the kernels and the host code were not optimized for working with FPGA. Some of these optimizations, such as the use of pipes or Altera Channels, will result in an overall reduction in run time outside of kernel performance. There also exists further scope for extracting more parallelism from the Hough transform code, the lack of which could also be affecting the performance currently. Alternatively, this could be indicative that this is the kind of problem that is suitable for different device architectures.

5. Conclusion

An evaluation of using OpenCL with FPGA has been completed. An implementation of a Data Acquisition system was attempted along with an implementation of an algorithm for acceleration. The idea behind implementing a DAQ system was to explore the possibility of using OpenCL for more than just acceleration. Many of the design elements needed to realize a DAQ system in OpenCL already exist, mostly as FPGA vendor extensions. Some of these extensions are also going into subsequent versions of the OpenCL specification. However a small number of elements are also missing, preventing one from fully realizing a complete DAQ system today. Since these missing elements have simple feasible solutions, they could also be implemented if the FPGA tool vendors so desire. Hence it might soon be possible to implement a complete DAQ system on FPGA using OpenCL. It still remains to be seen how such a system would perform compared to a custom implementation in VHDL/Verilog, but there definitely exists a case for OpenCL in this application due to the massive productivity gain and ease of use it offers. Also, since this is a software-based development flow, this enables even non FPGA experts to design, debug and maintain the code. While this approach relies on vender specific extensions to a large extent, portability of the code is not a major concern as these kind of designs cannot be realized on other devices, such as GPUs and other accelerators, anyway.

As for using OpenCL to accelerate algorithms on FPGA, in the test case the FPGA performed better in one instance and much worse in the other. Further work is necessary to determine the exact reason. An optimized implementation for FPGA would be a better measurement of the performance but irrespective of whether the performance is higher or lower than other devices, one thing that cannot be denied is that OpenCL makes exploiting FPGAs for acceleration as easy as exploiting GPUs. That is a long way from the days of painstaking efforts to create a cycle accurate HDL design, functionally verifying it, debugging the design errors and fixing the timing violations to realize a working system. Even if FPGAs lag behind other devices in terms of raw performance figures, which are not always the case, they are usually still better when the metric of comparison is efficiency, as defined as Performance/Watt. For example, both these devices under comparison are of the 28nm process generation. The maximum power consumption of the FPGA is about 25W and the actual consumption could be much less depending on the design. The same figure for the GPU is 195W. This order of magnitude power advantage usually translates into better Performance/Watt figures for the FPGA. Extracting more parallelism from the algorithm, creating an FPGA optimized implementation, investigating the huge drop in performance for some kernels and also accurate power profiling of the designs could be the direction of future work.

6. References

[1] Srikanth S 2014 Dynamically Adaptive Header Generator and front-end source emulator for a 100 Gbps FPGA based DAQ *Real Time Conf. (RT), 2014 19th IEEE-NPSS.* 1–4

[2] https://en.wikipedia.org/wiki/Hough_transform

[3] https://www.khronos.org/opencl/

[4] Barczyk A, Dufey J-P, Gaspar C, Gavillet P, Jacobsson R, Jost B, Neufeld N, and Vannerem P 2014 The new LHCb trigger and DAQ strategy: a system architecture based on gigabit-ethernet *IEEE Trans. Nucl. Sci.* **51** 456–460

[5] Dufey J P, Frank M, Harris F, Harvey J, Jost B, Mato P and Mueller E 2000 "The LHCb Trigger and Data Acquisition System" *IEEE Trans. Nucl. Sci.* **47** 1–5

[6] Ebert M 2014 Parallel Hough transform for track detection in LHCb's VELO Pixel detector *CERN-STUDENTS-Note-2014-201* See https://cds.cern.ch/record/1756405

[7] Altera 2013 Implementing FPGA Design with the OpenCL WP-01173-3.0

[8] Altera 2015 Altera SDK for OpenCL Programming Guide OCL002-15.0.0

[9] Hunter T M, Denisenko D, Kannan S, Bold J M, Thippabathini P and Dusel P P E 2014 FPGA Acceleration of Multifunction Printer Image Processing using OpenCL WP-01223-1.0