OpenCL kernels on FPGA architecture

www.exact-lab.it

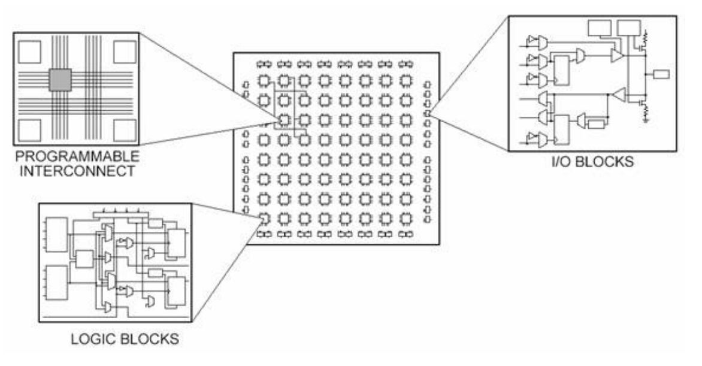
Authors: *Paolo Gorlani, Willi Menapace, Giuseppe Piero Brandino, Stefano Cozzini.*

This poster reports on the *eXact-lab srl* activities within the ExaNeSt project (`www.exanest.eu`) on the FPGA exploitation on selected scientific applications.

One of the tasks of eXact-lab, within the ExaNeSt project, is to evaluate the computational performance of the FPGA devices on the ExaNest board. Each board is equipped with 4 FPGAs, 3 of which are available for computation and 1 is reserved for network management. Using Xilinx tools such as Vivado HLS, scientific OpenCL kernels can be synthesized, allowing the FPGAs to be used as accelerators. The low power consumption of FPGAs compared to GPUs and the possibility for deep level of optimization, are expected to provide high level of computational efficiency.

In this poster we will present our preliminary results obtained in porting openCL kernel on the FPGA architecture selected by the Exanest project.

FPGA



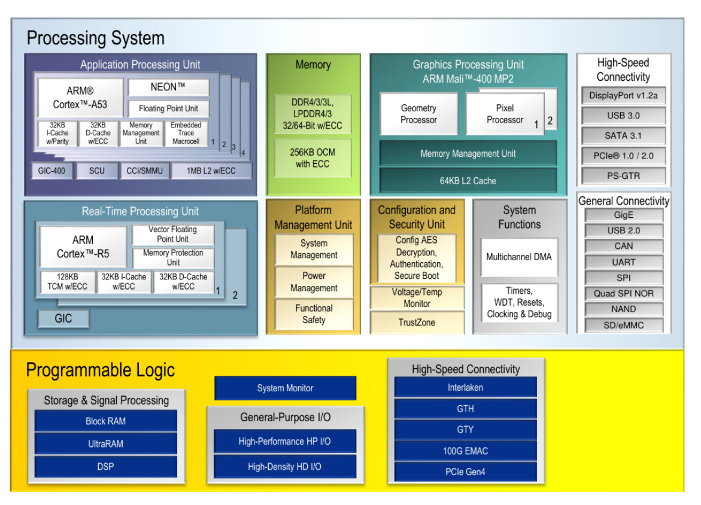
Field **P**rogrammable **G**ate **A**rrays are semiconductor devices that are based around a matrix of configurable logic blocks connected via programmable interconnects. This kind of devices make possible to develop compute functionality directly at the silicon level.

Our goal has been to synthesize OpenCL kernel functionality in a functional block into the FPGA logic, and to evaluate the computing performance.

Xilinx Multiprocessor System-on-Chip

A Multiprocessor System-on-Chip is a system-on-a-chip which uses multiple heterogeneous processors.

We conduct our tests using a *Xilinx Zynq Ultrascale+ XCZU9EG* chip, which has been equipped with 2 GByte of DDR4 memory. This chip is divided in two parts. The first is the processing system, among the others, it contains the ARM cores, the DDR subsystem. The second is the programmable logic, which contains the FPGA.



In order to exploit the full capabilities of the system, it is fundamental to understand the way the functional unit developed into the FPGA are connected with the processing system which includes the DDR memory subsystem.

These communications are managed by the AXI4 protocol.

AXI4 protocol

AXI4 is an on-chip interconnect specification for the connection and management of functional blocks in a System-on-Chip. We have employed two types of AXI4 buses/interfaces:

- *AXI4* for high-performance memory transactions. It allows burst of up to 256 data transfer cycles with just a single address phase. We use it to connect the functional blocks to the DDR memory.
- *AXI4-Lite* for simple, low-throughput, single memory transaction. We use it to control and to set the status of the functional blocks.

OpenCL kernel synthesis

In order to synthesize, implement and run on the FPGA an OpenCL code, we need to perform three main steps by means three software tools. We briefly describe the procedure in the following subsections.

Step 1. High Level Synthesis

The first step is done using *Vivado HLS*. It performs the High Level Synthesis of existing codes written in C, C++ and OpenCL. It produces an RTL design which have the same functionality of the considered coded function. RTL stands for Register-Transfer Level, a design abstraction that models a digital circuit.

Vivado HLS transforms an OpenCL kernel in a functional block having two interfaces:

- an AXI4-Lite interface: it is used to control the functional block and to pass all the kernel arguments;
- an AXI4 interface: it lets the functional block to access the DDR memory.

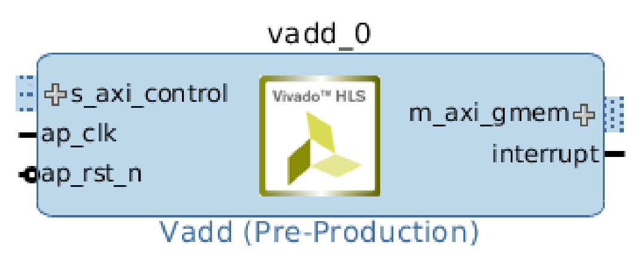


Figure 1: Synthesized functional block.

Vivado HLS provides various pragma in order to optimize the code synthesis. The output of this stage is a functional block (Figure 1), which needs to be integrated within the processing system.

Step 2. System design

The second tool is *Vivado*. It creates project designs (Figure 2), which integrates the RTL produced in the previous step with the rest of the system.

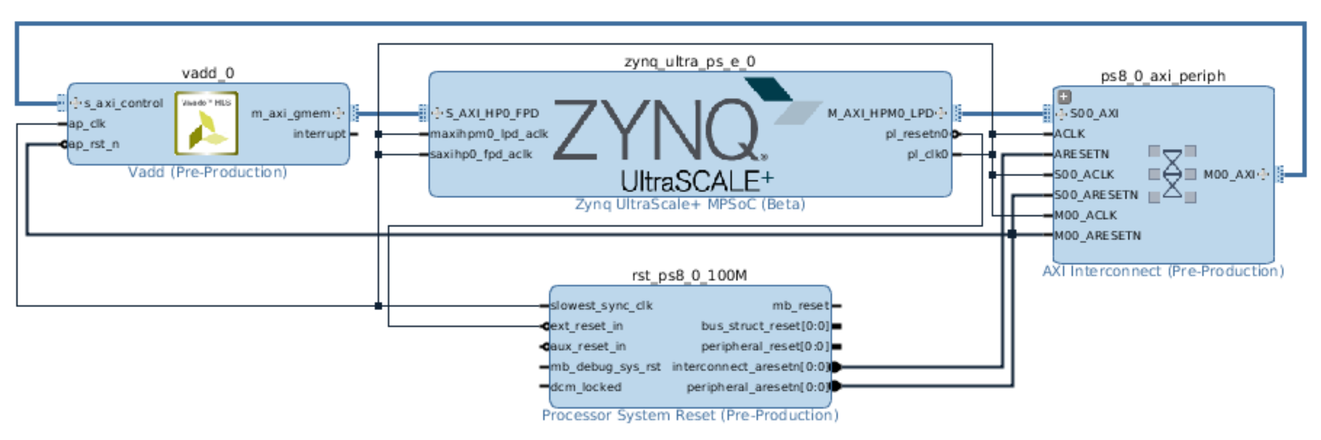

vadd_0
zynq_ultra_ps_e_0
ZYNQ UltraSCALE+
Zynq UltraScale+ MPSoC (Beta)
ps8_0_axi_periph
AXI Interconnect (Pre-Production)
rst_ps8_0_100M
Processor System Reset (Pre-Production)

Figure 2: Project design.

After various internal steps, *Vivado* produces a bitstream, which can be loaded into the FPGA.

Step 3. Software development

The last tool is performed by means of *Xilinx SDK*. It is a software development environment based on Eclipse which makes possible to develop, debug and test the executable running on the ARM, which controls the work of the functional block build in the FPGA logic.

Results

We report our preliminary results on three simple, yet meaningful test cases, in order to evaluate different features of our hardware.

Simple array summation

First of all we have implemented a simple array addition. This is usually a bandwidth bounded operation. In this way, we want to study the best approach to exploit the available bandwidth of our device.

We tested the bandwidth varying the number of functional blocks used and the FPGA operative frequency. Furthermore, we measure the bandwidth with and without burst memory transactions allowed by the AXI4 protocol.

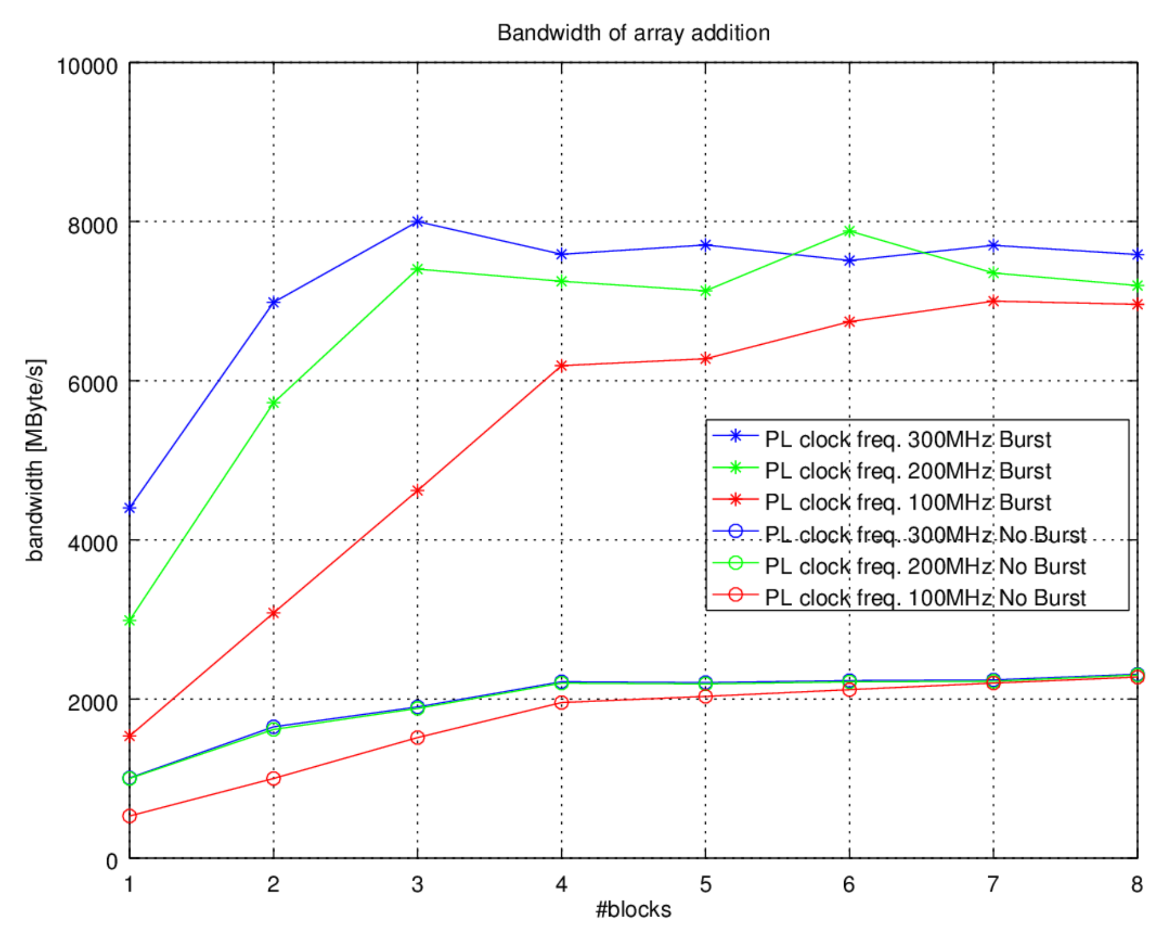


Figure 3: Measured bandwidth

Figure 3 shows the results. Burst memory transaction are fundamental to achieve good performance. The peak bandwidth in this case is around 7.8 GByte/s. Without burst the peak performance drops to 2.1 GByte/s.

Matrix multiplication

The second test case refers to the implementation of the matrix multiplication. We implemented a $O(n^3)$ block algorithm (written in OpenCL) best suited for GPUs. We compared the energy efficiency of the FPGA against two NVIDIA GPUs: the GTX 1080 for single precision floating point numbers, and the K20 for double precision floating point numbers. Results are shown Table 1.

	Single precision		**Double precision**	
	GPU	FPGA	GPU	FPGA
Performance [FLOPS]	5330	9.84	507	4.15
Power consumption [W]	165	7.23	110	7.58
Efficiency [FLOPS/W]	32.3	1.36	4.61	0.547

Table 1: Matrix multiplication results

Not surprisingly GPUs are two order of magnitude faster than FPGA implementation such large performance gap reduces considerably (one order of magnitude) when we look at the energy efficiency. It must be noted, however, that the implemented algorithm has been designed keeping in mind the architecture of GPUs and not the FPGA ones.

Force compute in Molecular Dynamic

We are porting an OpenCL kernel, which compute the forces in a Molecular Dynamic code. This kernel performs a lot of single memory access in order to get the data of close particles. This is the main bottleneck for FPGAs and GPUs. We tested various force kernel implementations playing Vivado HLS pragma. Table 2 reports our best results so far.

	GPU	FPGA
Execution Time [ms]	10	610

Table 2: Compute force execution times

The GPU is an NVIDIA K20. We can observe that performance penalty is reduced by a factor of ten with respect to the previous case. We are currently exploring the way to reduce the huge bottleneck due to the single memory transfers.

Conclusions

We ported successfully three different kernels and compared the performance against GPU devices. Even if, our results are preliminary, we learned some some key facts on design code for the FPGA.

- It is fundamental to use the pragmas available in Vivado HLS in the OpenCL kernel. The main goal on the FPGA is to design pipeline with a small initiation interval, in order to provide an high throughput.
- The memory transactions need to be performed in a burst in order to be fast.
- Accessing the memory in single transaction is a big performance limiter.